UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Esquivación de objetos en vehículos autónomos mediante técnicas de machine learning

Autor/es

DAVID VILLOTA MIRANDA

Director/es

JAVIER RICO AZAGRA y MONTSERRAT GIL MARTÍNEZ

Facultad

Escuela de Máster y Doctorado de la Universidad de La Rioja

Titulación

Máster Universitario en Ingeniería Industrial

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

Trabajo de Fin de Máster

ESQUIVACIÓN DE OBJETOS EN VEHÍCULOS AUTÓNOMOS MEDIANTE TÉCNICAS DE MACHINE LEARNING

Autor

David Villota Miranda

Tutores: Javier Rico Azagra, Montserrat Gil Martínez

MÁSTER:
Máster en Ingeniería Industrial (852M)

Escuela de Máster y Doctorado

UNIVERSIDAD DE LA RIOJA

AÑO ACADÉMICO: 2019/2020

Contenidos

Lista de imágenes

1. Resumen

El aprendizaje automático (*machine learning*) es capaz de dotar a los vehículos autónomos de cierta inteligencia para sus funciones de navegación. En este caso, se utiliza una red neuronal (RN) que, a partir de las imágenes capturadas desde un vehículo móvil, determina los puntos objetivo donde la probabilidad de colisión sea la menor posible. Estos se derivan a la arquitectura de control de guiado autónomo.

Para el adecuado entrenamiento de la red se precisa gran cantidad de información: imágenes de paisajes diversos que han de ser capturadas desde el vehículo en movimiento. En este sentido, y para salvaguardar la integridad del drone, los entornos virtuales utilizados en videojuegos serán de gran ayuda para generar imágenes sintéticas. En particular, se ha utilizado el motor gráfico Unreal Engine, en combinación con el simulador Airsim, que embebe el movimiento del drone dentro del paisaje virtual y permite configurar el formato de adquisición de imágenes. Después, las imágenes sintéticas son pre-procesadas para su uso por los algoritmos de visión artificial desarrollados en Python.

Para generar los datos de entrenamiento, se parte de las imágenes anteriores cuyos píxeles codificarán la profundidad y por tanto, se obtendrá un indicador de la probabilidad de colisión. Después se han desarrollado tres algoritmos de procesado de imágenes, que automatizan el modo de obtener el punto (coordenadas X-Y) óptimo o de menor probabilidad de colisión, denominado *ground-truth*.

Con las imágenes obtenidas y el *ground truth* generado, se procede a entrenar una red neuronal convolucional basada en la arquitectura ResNet-8 modificada en su última capa para poder generar dos salidas (coordenada X-Y).

Se estudian los diferentes parámetros de entrenamiento para optimizar el tiempo y la tasa de aprendizaje, dada la limitación de recursos computacionales de un ordenador personal. A lo largo del entrenamiento se realizan un seguimiento de la función de error para evitar un sobre ajuste a los datos de entrenamiento

(*Overfitting*) y de esta manera escoger la etapa (iteración del entrenamiento) que mejor comportamiento presente.

Se propone también una arquitectura de control en cascada que, con la información de coordenadas óptimas y la probabilidad de colisión genera las referencias de control para los lazos internos del drone.

Las funciones de preprocesado de imágenes, la generación del punto objetivo por la red neuronal y la realimentación del lazo de control, se deben ejecutar de forma paralela. Esto será posible gracias a ROS (*Robot Operating System*), su estructura nodal permite el intercambio de información de manera continua entre los tres grandes bloques para que la aplicación pueda ejecutarse en tiempo real con una latencia más pequeña posible.

Finalmente se propone una solución hardware para implementar parte del sistema. Dos cámaras de bajo coste conectadas a sendas Raspberries Pi Zero adquieren las imágenes estereoscópicas y las canalizan calibradas a una Raspberry Pi 3 B+. Esta se encarga del almacenamiento de imágenes y la gestión de las comunicaciones. El resto de funcionalidades descritas anteriormente se realizan actualmente en una estación de trabajo.

1.1. Palabras Clave

Visión estéreo, Python, Raspberry Pi, ROS, Raspbian, CNN, Red Neuronal, Dataset, Abstracción.

2. Abstract

Machine learning is capable of providing autonomous vehicles with some intelligence for their navigation functions. In this case, a neural network (NN) is used which, from the images captured from a mobile vehicle, determines the target points where the probability of collision is as low as possible. These are derived to the autonomous guidance control architecture.

For the proper training of the network, a great deal of information is needed: images of diverse landscapes that have to be captured from the moving vehicle. In this sense, and to safeguard the integrity of the drone, the virtual environments used in video games will be of great help to generate synthetic images. In particular, the Unreal Engine graphic engine has been used, in combination with the Airsim simulator, which embeds the drone movement within the virtual landscape and allows you to configure the image acquisition format. Then, the synthetic images are preprocessed for use by the artificial vision algorithms developed in Python.

To generate the training data, it is based on stereoscopic images acquired with Airsim, and taken from the moving vehicle. In each pixel an intensity is coded, which indicates the depth and, therefore, there is an indicator of the probability of collision. Afterwards, three image processing algorithms have been developed, which automate the way to obtain the optimum point (X-Y coordinate) or one with the lowest probability of collision, called ground-truth.

With the images obtained and the ground truth generated, we proceed to train a convolutional neural network based on the ResNet-8 architecture modified in its last layer to generate two outputs (X-Y coordinates).

The different training parameters are studied to optimize the time and the rate of learning, given the limitation of computational resources of a personal computer. Throughout the training, the error function is monitored to avoid an over-adjustment

to the training data (*Overfiting*) and in this way choose the stage (training iteration) of best perfomance.

A cascade control architecture is also proposed that, with the optimal coordinate information and the probability of collision, generates the control references for the internal loops of the drone.

The image preprocessing functions, the generation of the target point by the neural network and the feedback control loop, must be executed in parallel. This is possible thanks to ROS (Robot Operating System). Its nodal structure allows the exchange of information continuously between the three large blocks so that the application can be executed in real time with as little latency as possible.

Finally, a hardware solution is proposed to implement part of the system. Two low-cost cameras connected to Raspberries Pi Zero paths acquire stereoscopic images and channel them calibrated to a Raspberry Pi 3 B +. This is responsible for image storage and communications management. The rest of the functionalities described above are currently carried out in a workstation.

2.1. Keywords

Stereo vision, Python, Raspberry Pi, ROS, Raspbian, CNN, Neural Network, Dataset, Abstraction.

3. Agradecimientos

A todos los profesores que directa o indirectamente me han ayudado en este proyecto. Especialmente a Javier Rico y Montse Gil por su paciencia y confianza durante el desarrollo del proyecto.

Por último a toda la gente que me ha acompañado durante estos dos años en especialmente a mis compañeros que también me han apoyado en el proyecto. Y por supuesto a mi familia.

.

ESQUIVACIÓN DE OBJETOS EN VEHÍCULOS AUTÓNOMOS MEDIANTE TÉCNICAS DE *MACHINE LEARNING*

MEMORIA

4. Memoria

4.1. Antecedentes

Actualmente la navegación autónoma esta adquiriendo una gran importancia y se considera una de las hazañas a conseguir por el ser humano. No obstante la creación de sistemas robustos capaces proporcionar la seguridad necesaria requerida, hace incurrir en sensórica adicional y por lo tanto en un coste elevado.

Actualmente gracias al gran crecimiento que el "Machine learning" ha experimentado en estos últimos años, se están investigando nuevas vías enfocadas a abaratar costes.

Un claro ejemplo es la red convolucional desarrollada por la ETH (Escuela Politécnica Federal de Zurich) en Suiza, que con muchas limitaciones, consigue conducir un dron de manera autónoma esquivando objetos.

No obstante, para llegar a este resultado final, se debe encarar un proceso de selección del hardware a utilizar, entorno al cual, también ha habido muchos avances. Se han desarrollado controladores que soportan visión esterescópica como el CM3 y procesadores especiales para la implementación de redes neuronales.

En este TFM se tratará de buscar una solución capaz de llevar a cabo todo el procesado de imágenes de una manera fluida, además de ser ligera para su futura implementación en UAVs. La solución en este caso esta basada en Raspberry.

4.2. Objeto del trabajo

El objeto de este trabajo es el desarrollo de algoritmos capaces de identificar objetos, reales o sintéticos, probables de colisión, con el fin de ser esquivados por un vehículo en movimiento. Para ello se deberá diseñar o adaptar una red neuronal y su entorno en ROS, de cara a la implantación en un sistema real.

Los datos de entrenamiento de la RN, se compondrán de imágenes sintéticas generadas con *Unreal Engine* junto a *Airsim*, y el *ground truth* generado mediante algoritmos de esquivación.(el ground-thuth es la coordenada X-Y de la imagen con menor probabilidad de colisión)

La RN será capaz de identificar de manera automática el área de una imagen que no tenga probabilidad de colisión. Este área será representado mediante un punto de coordenadas [X,Y], posteriormente utilizado para generar las referencias de control de guiado enviadas al UAV.

También se creará un prototipo hardware *"low cost"* para adquirir imágenes estereoscópicas basado en tecnología Raspberry.

En la imagen siguiente se puede observar gráficamente la utilidad final de este proyecto. Desde la adquisición de imágenes mediante hardware hasta su procesamiento y generación de referencias para el control de guiado en el entorno ROS, se describirán a continuación los objetivos específicos.

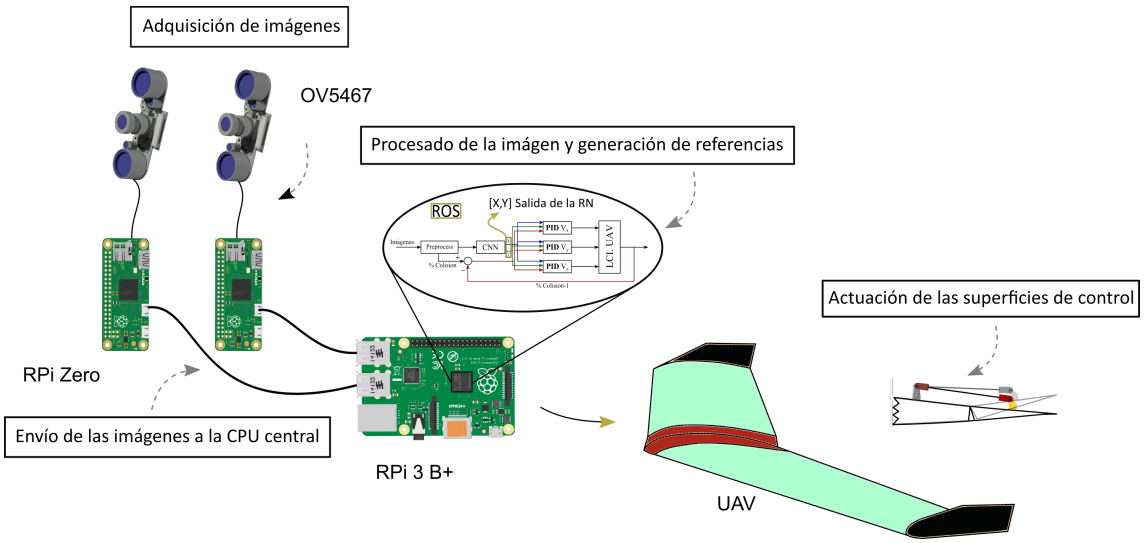


Figura 1: Utilidad del proyecto

4.2.1. Objetivos específicos

Para entender bien cada uno de los objetivos, se identificarán cada uno de ellos en la imagen siguiente donde aparecen todos los elementos que forman parte de la solución.

1. Selección de la arquitectura más eficiente para la red neuronal.
2. Configuración de ROS (*Robot Operating System*) y desarrollo en este sistema operativo de funciones para la gestión de la información generada y su tratamiento.
3. Implementación de la red neuronal y su entorno apropiado para su correcto funcionamiento en ROS.
4. Selección del hardware apropiado para la adquisición y procesado de imágenes.
5. Seleccionar y diseñar las comunicaciones de hardware pertinentes.
6. Implementación de la API Airsim en el software de diseño gráfico Unreal World.
7. Generación y procesado de los datos de entrenamiento para la Red neuronal.
8. Diseño de algoritmos para la generación de *"ground truth"* automático y selección del óptimo.

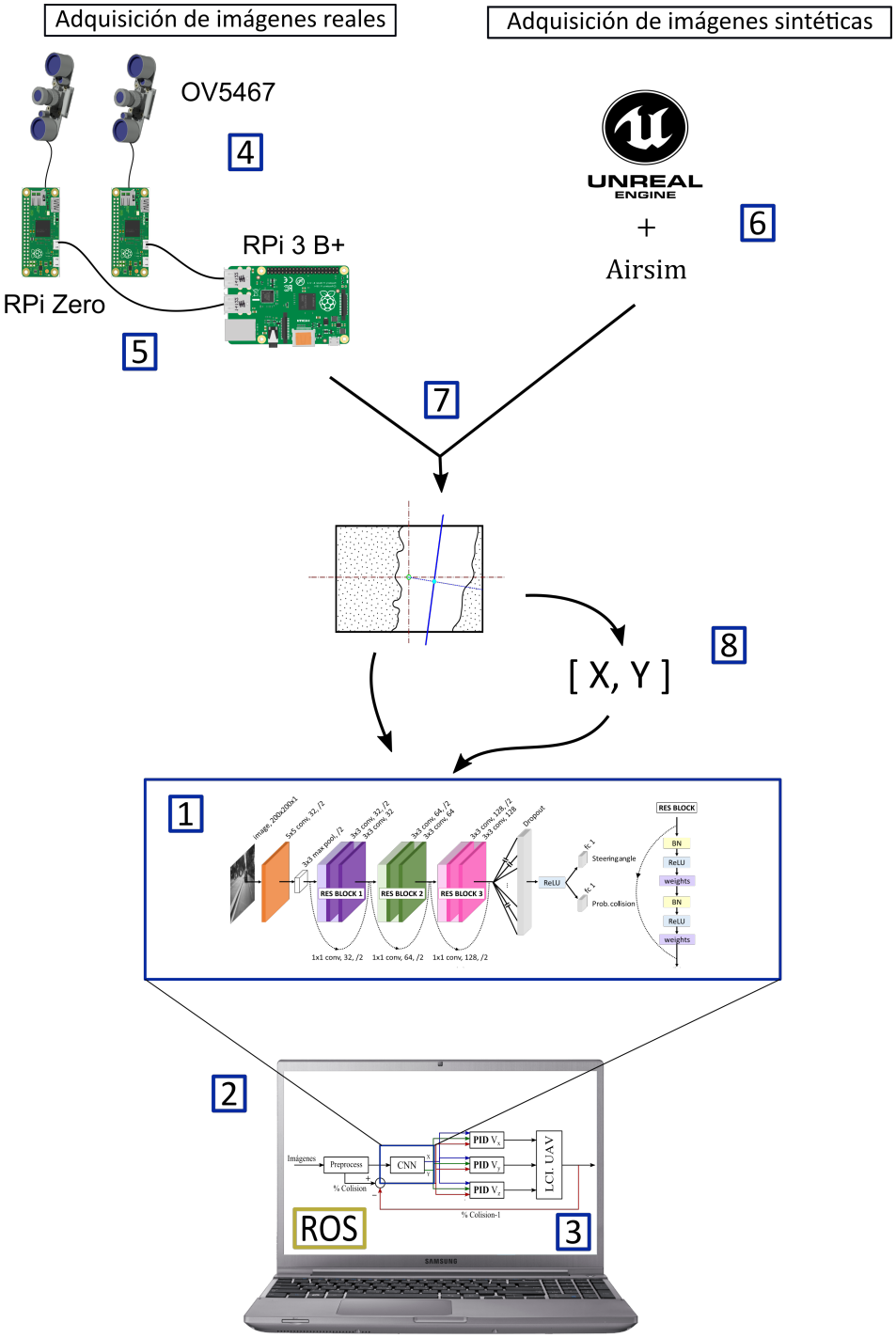


Figura 2: Secuencia de acciones a realizar

4.3. Referencias

4.3.1. Bibliografía web

- raspberryjamberlin.de
- pymotw.com/2/threading/
- desertbot.ios
- microsoft.github.io/AirSim/
- github.com/Jaeyoung-Lim/rosppm
- https://stackoverflow.com/
- http://rpg.ifi.uzh.ch/dronet.html
- https://www.iberdrola.com/innovacion
- https://github.com/tensorflow/tensorflow
- https://keras.io/
- https://wiki.debian.org/udev
- https://towardsdatascience.com
- http://cs231n.github.io/convolutional-networks/
- https://algorithmia.com/blog/introduction-to-optimizers
- https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html
- https://es.mathworks.com/discovery/stereo-vision.html
- https://www.ticbeat.com/tecnologias/

4.3.2. Artículos bibliográficos

1. Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza (2018). DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters*, *3(2)*.

2. M.Zanin, S. Messelodi, C.M. Modena (2004). DIPLODOC - A labelled road sequence taken from an in-vehicle stereo camera

3. Sashank J. Reddi, Satyen Kale & Sanjiv Kumar (2018). ON THE CONVERGENCE OF ADAM AND BEYOND. *ICLR 2018 Conference*

4. Villota Miranda, David; Gil Martínez, Montserrat; Rico Azagra, Javier (2018). Longitudinal control of a fixed wing UAV. *XXXIX Jornadas de Automática*

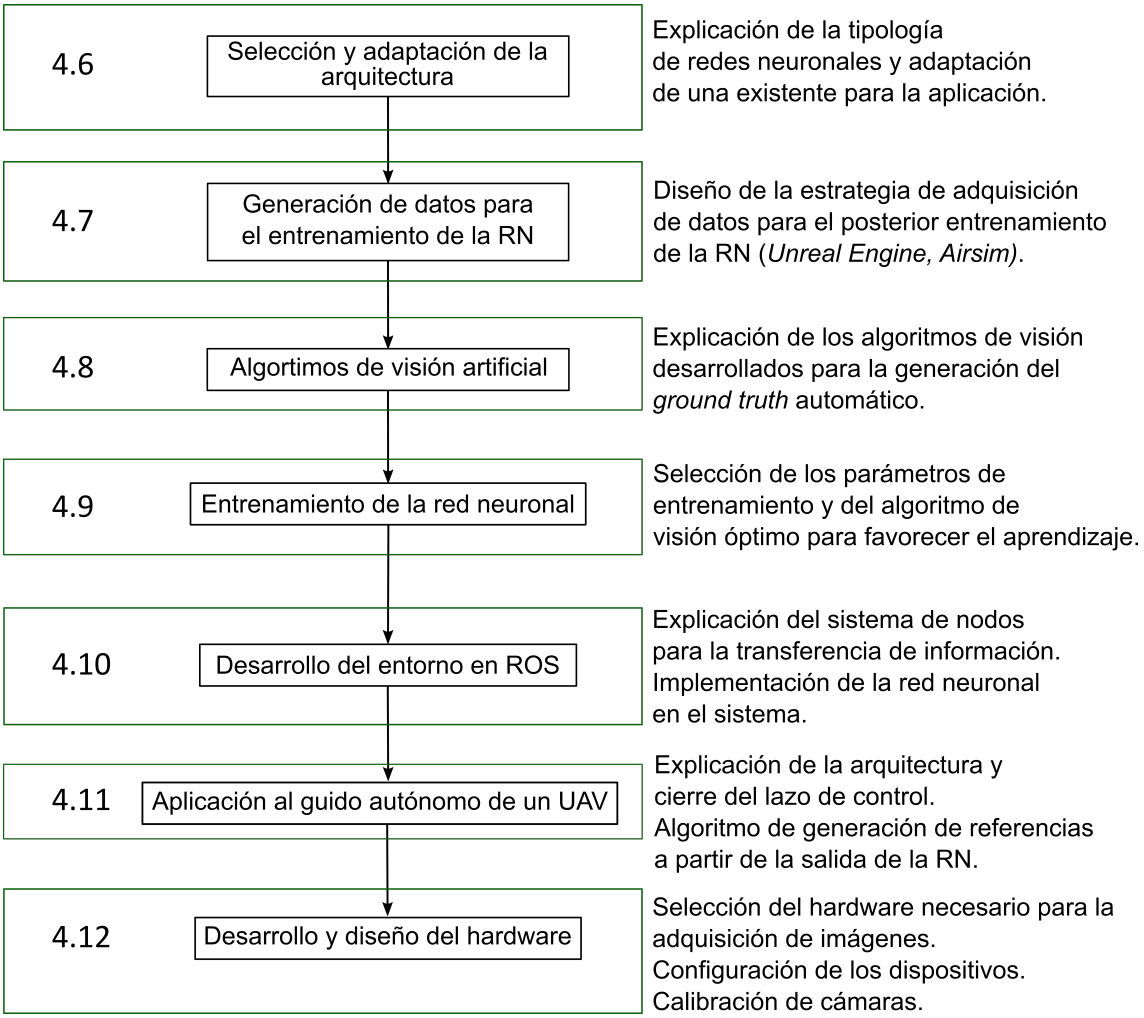
5. Zifeng Wu, Chunhua Shen, Antonvan den Hengel (2019). Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. *ELSEVIER Pattern Recognition*,Pages 119-133.

4.4. Definiciones y abreviaturas

- **Dataset:** Conjunto de datos utilizados para entrentar la red neuronal
- **Ground Truth:** Solucion correcta con la que se compara la salida de la red neuronal durante el entrenamiento.
- **CNN:***Convolutional neural network*
- **IA:** Inteligencia Artificial
- **RN:** Red Neuronal
- **API:** *"Application Programming Interface"*.Son un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas.
- **Visión artificial:** Disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador
- **Inteligencia artificial:** Combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano.
- **SO:** Sistema Operativo
- **Plug-in:** Aplicación que se relaciona con otra para agregarle una funcionalidad nueva generalmente muy específica.
- **ROS:** *Robotic Operating System*

4.5. Estructura

De acuerdo a los objetivos planteados, los resultados de este trabajo se van a presentar siguiendo el siguiente esquema.



4.6. Selección y adaptación de la arquitectura de RN

4.6.1. Introducción

En este apartado se explicará de manera general qué es la inteligencia artificial y sus principales aplicaciones.

Posteriormente se explicará la diferente tipología de redes neuronales de entre las cuales, tras entender la naturaleza de la problemática a solucionar, se escogerá la más apta para esta aplicación.

Una vez escogida se prestará atención a estudiar la funcionalidad y características de las capas que la componen con el fin de seleccionar la arquitectura que optimice el aprendizaje. En este caso se ha partido del trabajo realizado por la ETH Suiza en su proyecto *"Dronet"*, sobre el que se realizarán las modificaciones pertinentes para adaptarla a esta aplicación.

4.6.2. Inteligencia Artificial

La inteligencia artificial es una combinación de algoritmos cuyo fin es reproducir en máquinas capacidades humanas. Por definición en la ciencia de la computación una máquina inteligente es un agente flexible que percibe su entorno y actúa en consecuencia para mejorar sus probabilidades de éxito.

El término inteligencia artificial se empezó a utilizar en 1956, pero es actualmente cuando se esta popularizando debido al incremento en los volúmenes de datos, algoritmos avanzados, y mejoras en el poder de cómputo y el almacenaje. Es considerada una de las ramas de la ciencia computacional encargada de estudiar modelos de cómputo.

Según los expertos en el campo de la computación *Stuart Russell* y *Peter Norvig*, existen varios tipos de inteligencia artificial:

- **Sistemas de aprendizaje humano** Es el caso de las redes neuronales o machine learning. Tratan de automatizar la toma de decisiones y la resolución de problemas.
- **Sistemas de actuación humano** Realizan tareas físicas como la realizarían los humanos.
- **Sistemas de pensamiento racional** Imitan el razonamiento humano en respuesta a una serie de estímulos.
- **Sistemas de actuación racional** Imitan de manera racional el comportamiento humano.

Aplicaciones

La inteligencia artificial es una herramienta muy versátil y actualmente se está trabajando en ampliar sus límites de implantación.

- **Agrícolas.** Plataformas específicas que, por medio de análisis predictivos, mejoran los rendimientos agrícolas y advierten de impactos ambientales adversos.
- **Comercial.** Posibilita hacer pronósticos de ventas y elegir el producto adecuado para recomendárselo al cliente. Empresas como Amazon utilizan robots para identificar si un libro tendrá o no éxito, incluso antes de su lanzamiento.
- **Educación.** Permite saber si un estudiante está a punto de cancelar su registro, sugerir nuevos cursos o crear ofertas personalizadas para optimizar el aprendizaje.
- **Sanidad.** Ya existen chatbots que nos preguntan por nuestros síntomas para realizar un diagnóstico. La recolección de datos genera patrones que ayudan a identificar factores genéticos susceptibles de desarrollar una enfermedad.

4.6.3. *Machine Learning* (Aprendizaje automático)

Es una aplicación de la Inteligencia Artificial en la cuál se consigue la capacidad de aprendizaje automático. Se centra en el desarrollo de algoritmos informáticos que acceden y procesan una base de datos y aprenden automáticamente a través de ella.

Su aprendizaje se basa en la identificación de patrones complejos de millones de datos y su base es puramente estadística. En sus inicios en los años 50, se utilizaban únicamente para resolver sencillos problemas matemáticos como evaluar proximidad entre puntos o dirección de vectores. Actualmente la mitad de Internet utiliza este tipo de algoritmos.

Clasificación de Machine Learning

Hay dos clasificaciones posibles dependiendo de si el aprendizaje autónomo se encuentra supervisado o no. Es decir, que de alguna manera se le haga saber al sistema que lo que ha predicho es correcto o no.
Dentro de esta clasificación, nos podemos encontrar otras subcategorías tal y como se muestra en la siguiente imagen.

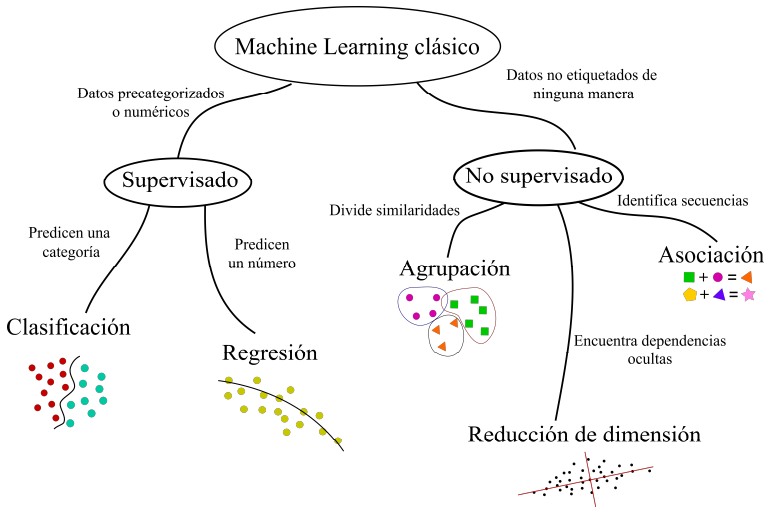


Figura 3: Mapa conceptual clasificación Machine Learning

Tal y como dice Michel Bowles en su libro *"Machine Learning in Python, essential techniques for predictive analysis"*, *"para solucionar el problema, el primer paso es entender los datos de los que se parten"*.

Objetivo del Machine Learning en este proyecto

En este proyecto se utilizará *Machine Learning* para que una red neuronal identifique de manera automática el área de un imagen donde sea baja la probabilidad de colisión entre el vehículo en movimiento y los objetos que esta visualizando según se desplaza. Este área será representado mediante un punto de coordenadas [X,Y], posteriormente utilizado para generar las referencias de control enviadas al UAV desde la estación de tierra (PC).

De esta manera, las imágenes entrarán en la red neuronal que dará como salida el punto objetivo. Este indicará en que dirección el dron deberá esquivar los obstáculos.

Para entrenarla se utilizarán imágenes sintéticas que previamente habrán sido procesadas con diferentes algoritmos de visión artificial para la generación del *"ground truth"*. Esto indica que el aprendizaje sera **supervisado**.

Por otro lado, la salida de la red neuronal no es un atributo booleano de la imagen. Son dos números decimales que indican la posición de la zona segura. Por lo tanto es un caso de **regresión**.

4.6.4. Tipologías de redes neuronales

PN: Preceptron Network

Es el modelo más simple y antiguo de neurona. Toma varias entradas, las suma aplica una función y arroja una salida.

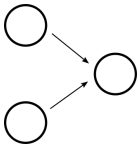


Figura 4: Arquitectura PN

FFN: Feed Forward Network

También bastante antiguas, sigue las siguientes normas:

- Todas las capas están totalmente conectadas unas con otras.
- La activación de las capas fluye desde las capas iniciales a las finales sin bucles hacia atrás. Es decir la activación de las primeras capas afectará de manera directa a las siguientes capas.
- Hay una capa entre la entrada y la salida.(Capas ocultas)

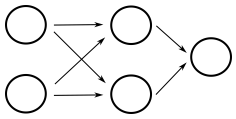


Figura 5: Arquitectura FFN

CNN: Convolutional Neural Network

En este caso las neuronas se correspondes con campos receptivos muy similares a las neuronas de la corteza visual primaria de un cerebro biológico. Por ello son las más utilizadas en procesamiento de imágenes, puesto que con pocos recursos es capaz de ofrecer buenos resultados.

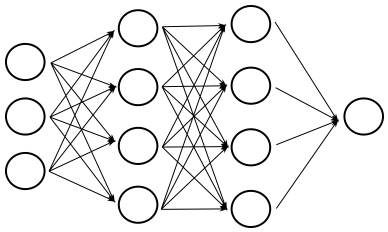


Figura 6: Arquitectura CNN

RNN: Recurrent Neural Network

Es una evolución de la FFN en la que algunas neuronas son realimentadas con su propia salida. Son utilizadas cuando decisiones tomadas en el pasado, pueden afectar a las del presente.

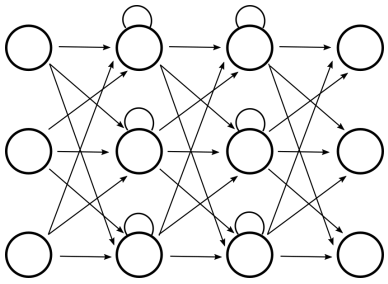


Figura 7: Arquitectura RNN

LSTM:Long/Short Term Memory

Este tipo de redes, incluye una celda de memoria que puede procesar datos. Es la versión avanzada de una RNN. Si la RNN puede procesar una palabra teniendo en cuenta el contexto en el que se encuentra, la LSTM puede procesar un fotograma de un vídeo en consecuencia con otros fotogramas anteriores

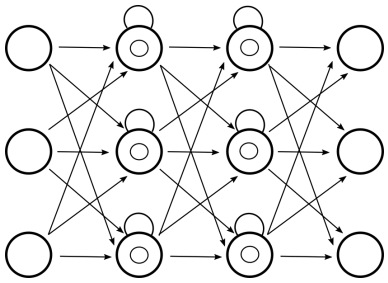


Figura 8: Arquitectura LSTM

Para esta aplicación la arquitectura que mejor se comportará será la red neuronal convolucional ya que sus múltiples capas de convolución permiten extraer de manera eficiente la información de las imágenes reduciéndola a valores numéricos.

4.6.5. ¿Cómo funciona una red neuronal convolucional?

Una red Neuronal convolucional es un algoritmo de *Deep Learning* que toma como entrada una imagen y arroja como salida información extraída de ese *input*. La arquitectura de una red Neuronal es análoga a los patrones de conexión de las Neuronas del cerebro humano y fue inspirada por la organización *"Visual Cortex"*.

Input

En este caso la imagen de entrada estará formateada en escala de grises, es decir sólo tendrá codificado el color blanco en sus píxeles con valor de 1 a 255. También es posible introducir otro tipo de codificaciones como RGB, HSV, CMYK, ETC.
En este caso será imágenes sintéticas en un gran porcentaje, generadas a partir *Unreal Engine* explicado más adelante. En menor cantidad imágenes reales descargadas de Internet. En ambos casos serán simulaciones de vuelos en diferentes entornos, bosque, costa, ciudad...

4.6.6. Función Loss

Son funciones matemáticas que miden cómo de bien la red neuronal está haciendo su trabajo con los pesos actuales. La elección de la función es muy importante para un correcto desempeño de la labor. Para este proyecto será interesante conocer MSE. Esta es la más común en problemas de regresión:

MSE.(*Mean Square Error*)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_{predicted} - X_{real}) \tag{1}$$

4.6.7. Capas utilizadas en la CNN

Convolutional layers. Estas capas tienen el objetivo de sintetizar la información proveniente de la imagen mediante la filtración de la entrada. De esta manera se consigue extraer la información de de la imagen. La primera capa suele aportar

información básica como bordes de siluetas y a medida que se van añadiendo capas en paralelo se consigue recopilar información de más alto nivel como texturas.

Pooling Layers. Son utilizadas para reducir la dimension espacial en una red convolucional. De esta manera se ahorra coste computacional y además reduce las posibilidades de que se produzca sobre aprendizaje (*"Overfit"*). Los principales elementos que definen la capa son las dimensiones iniciales del dato a procesar, el *"Stride"*, que es un escalar que indica el número de píxeles tras la capa y por último el Kernel size, que es una corrección numérica.
También suprime el posible ruido que pueda aparecer en la imagen. Las dimensiones finales serán definidas mediante:

$$H_2 = \frac{H_1 - K}{S} + 1 \tag{2}$$

$$Depth_{out} = Depth_{in} \tag{3}$$

Dropout. Es una técnica de regularización patentada por Google para reducir el sobre aprendizaje (*"Overfitting"*). El término *Dropout* viene del ingles *"Drop out"*, abandonar simplemente es administrar pesos aleatorios a neuronas de manera continua durante el aprendizaje. Es una manera muy eficiente de mejorar los modelos de redes neuronales

ReLu. En inglés *Rectified linear Unit* Es una función de activación definida de la siguiente forma:

$$f(x) = ln(1 + e^x) \tag{4}$$

Donde *x* es la entrada de una neurona. es una función rampa, su análogo electrónico es un rectificador de media onda. Es de las funciones de activación más comunes en Deep Learning.

4.6.8. Optimizadores

Durante el entrenamiento se cambian los pesos para intentar minimizar la función loss y hacer las predicciones lo más precisas posible. La cuestión es, cómo, cuándo y cuánto se deben cambiar los pesos para conseguir que la predicción mejore. Para eso se emplean los optimizadores, son los encargados de modificar los pesos según la salida de la función loss. Dicho de otra manera, la función loss es la ”guía” que indica al optimizador si está modificando los pesos correctamente. Existen diferentes conceptos a entender antes de explicar el algoritmo de optimización utilizado.

- **Gradiente descendiente**. (*Gradient Descent*)

  Es el más utilizado en machine learning, es robusto, rápido y flexible. Su manera de funcionar es muy sencilla.

  Cuantifica lo que un pequeño cambio en cada peso afectará a la función loss.A continuación ajusta cada peso individual basado en su gradiente y repite ambos pesos hasta reducir la función loss al mínimo.
  Lo más complejo de este algoritmo es entender los gradientes. Estos son una medida de cambio, matemáticamente hablando se corresponden con las derivadas parciales de la función loss con respecto a cada peso .

  Un problema derivado de este algoritmo surge de la aparición de mínimos locales. Estos mínimos pueden ocasionar un atrapamiento del algoritmo pensando que ya a alcanzado el mínimo absoluto. Para evitar esto se tendrá en cuenta el *Learning rate*.

- **Tasa de aprendizaje**.(*Learning rate*)

  Se trata de no realizar grandes saltos entre cambios de pesos para facilitar la búsqueda del mínimo absoluto. Una tasa de aprendizaje demasiado alta puede causar problemas de convergencia, mientras que si es demasiado pequeña, puede llevar a converger en mínimos locales.

Es básicamente un número que normalmente ronda el valor 0.001, este multiplica al gradiente para escalarlo. De esta manera existe la certeza de que los cambios llevados a cabo en los pesos son muy pequeños.

- **Regularización**

  Esta enfocado a prevenir el *"Overfitting"*. El overfitting significa que el modelo funciona muy bien sobre los datos utilizados en el entrenamiento pero no en otros datos ajenos.

  El objetivo de la regularización es penalizar aquellos pesos cuyo valor es demasiado alto. De esta manera los pesos serán penalizados si la predicción es incorrecta y si su valor es demasiado grande.

Los optimizadores utilizados en este algoritmo serán los siguientes:

Adam,

Este algoritmo utiliza una combinación de fracciones de gradientes anteriores para calcular el gradiente actual. Esto hace que sea muy sensible a la tasa de aprendizaje.

Es bastante innovador y se esta expandiendo muy rápido en el mundo de las redes neuronales.

AdamMax,

Es una variación del Adam basado en reducciones infinitesimales. Presenta mejor índice de convergencia y es menos sensible a la tasa de aprendizaje que el anterior. No obstante esto ocasiona mayores casos de *Overfitting*.

4.6.9. Modificaciones en Dronet

Dronet es una red neuronal creada en la ETH (Universidad Politécnica Federal de Zurich) con una arquitectura convolucional y una modificación en la última capa para

permitir a la red neuronal tener dos salidas diferentes. Con el fin de mejorar el coste y el tiempo computacional, ambas comparten todas las capas de la red a excepción de la última que tiene una capa para cada salida, en el caso original *"steering"* y *"collision"*.

En un principio se trató de eliminar la función sigmoidea de la variable de colisión de manera que ambas bifurcaciones sean exactamente iguales y únicamente se diferencien en los pesos. Tras correr varios entrenamientos se concluye que esta arquitectura no es capaz de aprender los parámetros necesarios para poder desempeñar la tarea deseada. Esto lleva a eliminar la bifurcación en la última capa e implementar dos redes convolucionales simples cuyas salidas sean las coordenadas objetivo "X" e "Y".

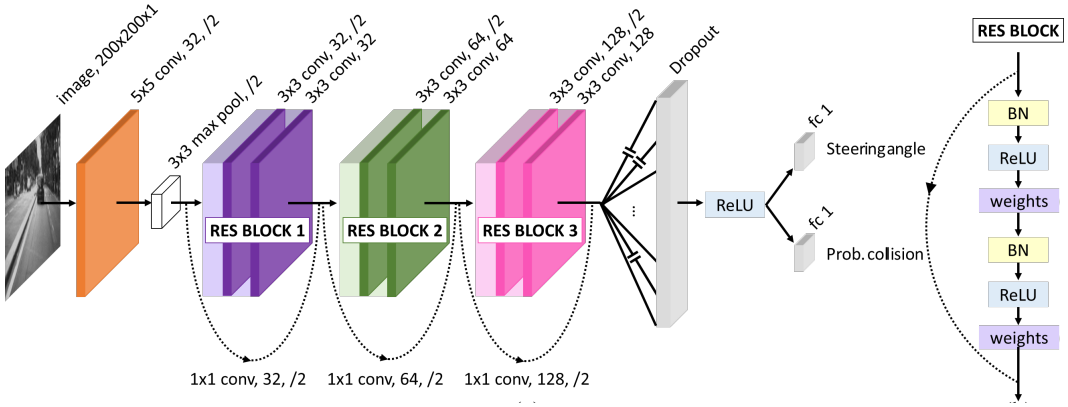


Figura 9: Arquitectura Dronet

La parte de la red neuronal compartida por las dos tareas se corresponde con una ResNet-8 seguida de un dropout de 0.5 y una ReLu no lineal. De esta forma se consigue aprovechar las ventajas de las arquitecturas ResNet y a la vez se consigue solucionar que para solucionar el problema derivado de divergencia en la optimización. En la última fase de la red neurnal, después de la última ReLu ambas tareas dejan de compartir pesos y la arquitectura se divide en dos capas totalmente conectadas (*"fully connected layers"*). Ambas darán como salida un valor entre -1 y

1 que será transformado mediante la siguiente fórmula en la coordenada en imagen:

$$X_{final} = X_{salida} \cdot 200 + 200 \tag{5}$$

Dronet estaba preparada inicialmente para dar un valor de steering [-1,1] y un valor de colisión [0,1]. Para adaptarla a las necesidades del proyecto se ha eliminado la función sigmoidea de la capa de colisión de manera que amplíe el intervalo a [-1,1]. Entre otras adaptaciones también se ha modificado la función loss puesto que para cada variable tenían una función diferente *MSE* para la variable steering y *BCE* para la variable de colisión. Para este caso es necesario que ambas tengan la función loss MSE.

Finalmente se muestra el código con las modificaciones pertinentes.

```
def resnet8(img_width, img_height, img_channels, output_dim):

    img_input = Input(shape=(img_height, img_width, img_channels))

    x1 = Conv2D(32, (5, 5), strides=[2,2], padding='same')(img_input)
    x1 = MaxPooling2D(pool_size=(3, 3), strides=[2,2])(x1)

    x2 = keras.layers.normalization.BatchNormalization()(x1)
    x2 = Activation('relu')(x2)
    x2 = Conv2D(32, (3, 3), strides=[2,2], padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x2)

    x2 = keras.layers.normalization.BatchNormalization()(x2)
    x2 = Activation('relu')(x2)
    x2 = Conv2D(32, (3, 3), padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x2)

    x1 = Conv2D(32, (1, 1), strides=[2,2], padding='same')(x1)
    x3 = add([x1, x2])

    x4 = keras.layers.normalization.BatchNormalization()(x3)
    x4 = Activation('relu')(x4)
    x4 = Conv2D(64, (3, 3), strides=[2,2], padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x4)

    x4 = keras.layers.normalization.BatchNormalization()(x4)
    x4 = Activation('relu')(x4)
    x4 = Conv2D(64, (3, 3), padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x4)

    x3 = Conv2D(64, (1, 1), strides=[2,2], padding='same')(x3)
    x5 = add([x3, x4])

    x6 = keras.layers.normalization.BatchNormalization()(x5)
    x6 = Activation('relu')(x6)
    x6 = Conv2D(128, (3, 3), strides=[2,2], padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x6)

    x6 = keras.layers.normalization.BatchNormalization()(x6)
    x6 = Activation('relu')(x6)
    x6 = Conv2D(128, (3, 3), padding='same',
                kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(1e-4))(x6)

    x5 = Conv2D(128, (1, 1), strides=[2,2], padding='same')(x5)
    x7 = add([x5, x6])

    x = Flatten()(x7)
    x = Activation('relu')(x)

    X_axe = Dense(output_dim)(x)
    Y_axe = Dense(output_dim)(x)

    model = Model(inputs=[img_input], outputs=[X_axe, Y_axe])
    return model
```

4.6.10. Bibliotecas necesarias para la implementación de RN

Se tratará de explicar brevemente las bibliotecas principales que han sido necesarias para llevar a cabo el proyecto. Actualmente son las dos más conocidas en el campo de la inteligencia artificial.

Tensorflow

Es una plataforma end-to-end open source para machine learning.Contiene una entorno flexible de herramientas y librerías que permiten a sus usuarios construir fácilmente aplicaciones de machine learning.

Figura 10: Tensorflow

Fue originalemnte desarrollado por los investigadores de la organización *"Google Machine Learning Research"* para la investigación de *machine learning* y *deep neural networks*. Provee de APIs estables desarrolladas en Python y C++.

Keras

Keras es una API de alto nivel capaz de implementar y compilar Tendorflow o similares como CNTK O Theano. Esta totalmente desarrollada en Python y permite crear fácilmente redes neuronales, así como todas las operaciones previas de acondicionamiento de datos.

Figura 11: Tensorflow

Una de sus grandes ventajas es que es capaz de ejecutarse tanto en GPU como en CPU, Por lo que acerca la posibilidad de *jugar* con redes neuronales a todos los usuarios que tengan un ordendor sin requisitos especialemente difíciles de cumplir. Otra potencialidad es que es capaz de implementar redes tanto convolucionales como recurrentes, o ambas a la vez.

4.7. Generación de datos para el entrenamiento de RN

4.7.1. Introducción

Para que la red neuronal funcione correctamente, necesitará aprender sobre datos de experiencias previas.

Estas experiencias previas se utilizarán como datos de entrenamiento. La generación de estos es la etapa más complicada. Suele ser un proceso muy costoso en tiempo y recursos.

En este apartado se presentará un proceso de adquisición de imágenes sintéticas que facilita la creación de datos de entrenamiento reduciendo el problema navegar por un entorno virtual como si fuese un videojuego.

Para ello se utilizará ***Unreal Engine***, que será el motor gráfico a través del cual se gestionará todo el medio sintético, y **AirSim**, un Plug-in que permitirá configurar y personalizar cómo se van a guardar las imágenes. En la siguiente imagen se puede observar la funcionalidad de ambas.

La gran potencialidad de este motor gráfico junto a Airsim, es la generación automática de imágenes de entrenamiento sin necesidad de arriesgar la integridad del dron. Airsim tal y como se ha detallado anteriormente, permite el almacenamiento automático de diferentes tipos de imágenes que se detallarán más adelante junto a variables dinámicas como velocidades lineales y angulares.

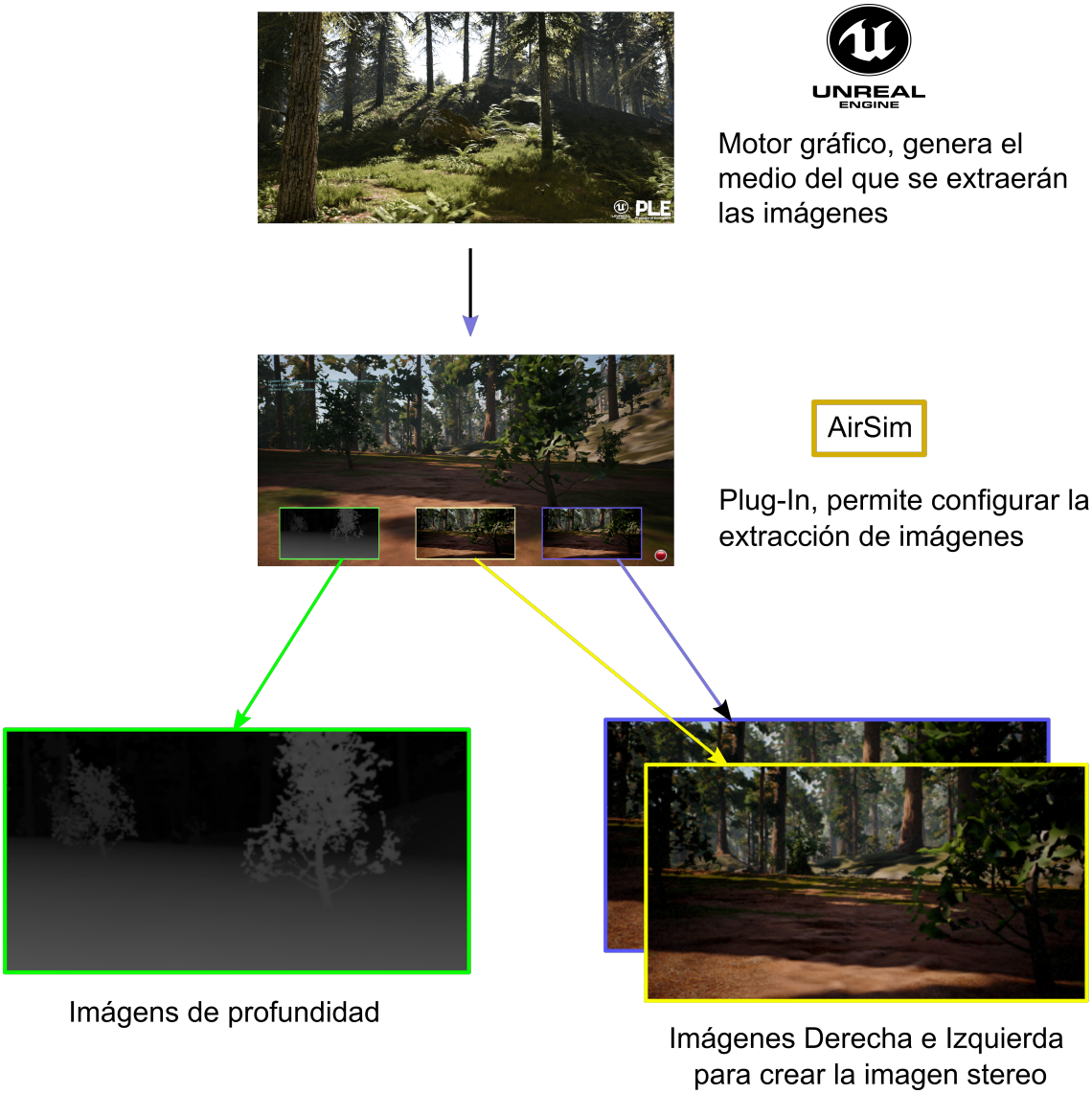


Figura 12: Funcionalidad de cada componente

4.7.2. *Unreal engine* motor gráfico

Unreal engine es un motor gráfico que nació en 2015 de manera gratuita. Pertenece a la compañía *Epic Games*. En un principio estaba destinada principalmente para desarrolladores, sin embargo cualquier usuario puede descargarlo gratuitamente. *Epic games* únicamente exige un ingreso del 5 por ciento de los primeros 3000 euros que se ganen fruto de la aplicación creada. Es un software muy versátil que permite la creación de videojuegos de todos los niveles de complejidad. Desde plataformas 2D a 3D.

Un motor gráfico debe ofrecer al programador una funcionalidad básica que permita a través de gráficos 2D y 3D, expresar y representar sus ideas en una pantalla de ordenador. Entre estas funcionalidades están la colisión física entre objetos, sonidos, música, etc. Todo esto permite que los desarrolladores no tengan que destinar gran cantidad de recursos y se puedan centrar en lo realmente importante que es su creación.

Dentro de las alternativas actualmente en el mercado, Unreal Engine 4 presenta unas capacidades gráficas impresionantes. Permite controlar hasta un millón de elementos en una misma escena. Entre otras características destaca la iluminación dinámica.

Para este proyecto se utilizarán paisajes virtuales gratuitos que se pueden descargar libremente desde la página oficial del software. De esta manera se pueden adquirir gran cantidad de secuencias de imágenes en ambientes muy diferentes. Para este proyecto en especial se han utilizado los siguientes paisajes:

- Sabana Africana
- Bosque montañoso
- Paisaje de costa tipo
- Paisaje residencial tipo

- Islas flotantes.

Como se puede observar, las imágenes que utilizadas provienen de paisajes muy variopintos que serían impensables de utilizar si la adquisición fuese manual.

Esto además permitiría el entrenamiento en medios inaccesibles como podría ser la superficie lunar, sin arriesgar ningún medio físico.

4.7.3. AirSim

Unreal Engine, permite la creación del entorno pero no la configuración de cómo se va a navegar a través de el, o de cómo se van a guardar las imágenes. Como se ha explicado antes, las imágenes destinadas al entrenamiento de la red neuronal, deben tener un formato característico que debe ser respetado. Para conseguir todo ello se utilizará AirSim.

Qué es Airsim

AirSim es un simulador para drones, automóviles y más, implementado en *Unreal Engine*. Es de código abierto, multiplataforma y admite hardware en bucle con controladores de vuelo populares como PX4 para simulaciones realistas física y visualmente. Está desarrollado como un complemento implementable en cualquier entorno de Unreal. Por lo tanto, como se ha comentado antes, se podrán crear paisajes customizados en los que ejecutar el simulador y ver como los prototipos o algoritmos se comportan en el mismo.

AirSim fue desarrollado con el fin de crear una plataforma de investigación en IA de cara a experimentar con algoritmos de *Deep Learning*, visión artificial y "Reinforced Learning" en vehículos autónomos. Con este propósito, Air-Sim también expone APIs de recuperación de datos y control de vehículos.

Control programático de Airsim

Esto permite al usuario interactuar con el vehículo en simulación mediante programación. Con ello se puede programar adquisición de imágenes, parámetros de control y parámetros de posición del vehículo. Permite programar en una gran variedad de lenguajes como C++, Python y Java.

4.7.4. Recopilación de datos de entrenamiento

Hay dos formas de generar datos de entrenamiento de AirSim para inteligencia artificial. La forma más fácil es simplemente presionar el botón de grabación en la esquina inferior derecha y automáticamente se empezará a crear un txt con información de la simulación como parámetros de velocidad aceleración, posición...

En el caso de que se haya programado adquisición de imágenes, se creará una carpeta automáticamente donde se guardarán las imágenes deseadas. Una mejor manera de generar datos de entrenamiento exactamente de la manera que desea es accediendo a las API. Esto le permite tener el control total de cómo, que, dónde y cuando desea registrar datos.

Tipos de imagen

- **Profundidad**. En este caso, los píxeles negros puro significarán una profundidad de 0 metros. Los píxeles blancos significarán una profundidad de 100 metros. En este caso, esta opción es la que se utilizará posteriormente.
- **Disparidad**. Se refiere a la diferencia de posición de la misma posición de pixel en un par de imágenes estéreo. Los objetos más cercanos presentarán una disparidad mayor, mientras que los más lejanos tendrán una disparidad muy pequeña.
- **Segmentación**. Es el proceso de dividir una imagen digital en varias partes objetos. El principal objetivo de este proceso es simplificar una imagen de cara a facilitar su análisis.

4.7.5. Creación de un medio sintético apto para Airsim

En primer lugar, debe de haber un medio diseñado sobre el cual se pueda implementar el PlugIn. La gran ventaja es que en este caso se puede crear un medio totalmente imaginario y surrealista, donde el algoritmo aprendería a volar autónomamente. En este caso, se partirá de diseños distribuidos de manera gratuita puesto que diseñar un medio nuevo lo suficientemente grande como para poder simular un dron en el interior es inabordable.

Para añadir el PlugIn al medio diseñado, es necesario añadir el directorio del mismo al script de construcción “build.bat”. De esta manera, tras ejecutar el .bat, se generará un archivo .sln (Extension de visual estudio). En él se deberán modificar los siguientes campos para habilitarlo:

- En el menú Edit -> Editor Preferences -> Search box -> CPU.

  Esto se hace con el fin de optimizar los recursos del ordenador. Al dejar de ser un proceso puramente gráfico procesado por la GPU, se debe cambiar el modo de procesamiento a la CPU que manejará con mayor fluidez las operaciones requeridas por Airsim.

- Build configuration -> Develop Editor & x64. Con el fin de habilitar el modo desarrollador que permite el uso de funciones avanzadas que incluye Airsim.

Una vez realizados estos pasos, desde la aplicación Unreal, se puede compilar el proyecto para comprobar que las funcionalidades de Airsim han sido añadidas.

4.7.6. Configuración de Airsim para la creación del Dataset.

Para este proyecto es necesario almacenar las imágenes con la profundidad codificada. Para la configuración de Airsim se utilizará Visual Estudio.

Desde este Script, se puede personalizar el modo de comportamiento. Es un código muy sencillo e intuitivo que permite configurar el tipo de imagen a almacenar y la posibilidad de añadir una subventana inferior donde poder ver en tiempo real la información deseada.

Para este proyecto se extrae una imagen codificada en .pfm que contendrá la profundidad, y por otro una imagen estéreo configurada con el mismo angular que las cámaras físicas para conseguir el mismo efecto.

A continuación se muestra la configuración de los campos *"Recording"* y *"Subwindows"*.

Figura 13: Airsim Interface

De esta manera cada vez que se pulse R durante la simulación se empezarán

```
"SettingsVersion": 1.2,
"SimMode": "ComputerVision",
"Recording": {
    "RecordOnMove": true,
    "RecordInterval": 0.05,
    "Cameras": [
{
        "CameraName": "0",
        "ImageType": 4,
        "PixelsAsFloat": true,
        "Compress": true
         }
    ]
},
"SubWindows": [
{
    "WindowID": 0,
    "ImageType": 4,
    "CameraName": "front_right",
    "Visible": true
       }
    ]
}
```

- Modo de funcionamiento
- Campo de grabación
- •Grabar a pesar de que el vehículo este parado
- •Fps
- •Tipo de cámaras
  - Nombre de la cámara
  - Tipo de imagen: Profundidad
  - Guardar en extensión pfm o jpg
  - Guardar las imágenes comprimidas
- •Subventanas
  - Identificador de ventana
  - Tipo de imagen a mostrar
  - Cámara escogida
  - Visible o no

a guardar las imágenes de manera automática y el interfaz tendrá la siguiente apariencia.

Estas imágenes son codificadas automáticamente bajo la extensión .pfm. Se utiliza esta extensión puesto que asegura que los carácteres y los glitches están correctamente escalados, lo cual es crucial en este tipo de trabajos tal como se explicará más adelante.

4.7.7. Preprocesado de las imágenes

Una vez se tienen todas las imágenes necesarias para entrenar lared neuronal, se debe llevar a cabo un preprocesado de las mismas para eliminar posibles defectos y transformarlas a un formato fácilmente comprensible para las librerías de Python.

Hay un fenómeno ocurrido durante la grabación de datos en Airsim, que debe ser tenido en cuenta. Cuando se está ejecutando la simulación durante una grabación, en muchas ocasiones se pueden producir colisiones con los objetos virtuales del entorno. Esto supondría atravesarlos, cosa que no está correctamente depurado en la API, la cual asigna valores desproporcionados a los píxeles en los que se ha dado el choque.

Por ello, es necesario procesar la imagen antes de introducirla en la red neuronal, este proceso se llamará *reescalado*. Mediante código:

- Se almacenará el valor más alto de toda la secuencia de vídeo y se hará la media de todos los píxeles. De esta maneta, si el valor máximo es un número arbitrario de veces mayor que la media, ese fotograma será automátiamente descartado
- Este proceso se repetirá de manera indefinida hasta que no se produzcan errores.

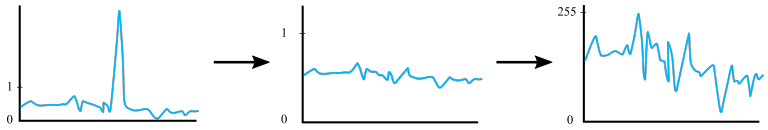


Figura 14: Flujograma de código

- A continuación se volverá a buscar el valor máximo pero esta vez para asignarle el máximo valor de codificación, en este caso 255, que significará el valor máximo de colisión

- Finalmente se reescalará mediante una simple proporción todos los valores que componen todos los fotogramas. Dando lugar a un vídeo sin ningún tipo de conflicto,perfectamente procesable por la red neuronal.

4.8. Algoritmos de visión artificial

4.8.1. Introducción

Para obtener buenos resultados en el entrenamiento de la red neuronal será necesario un conjunto de datos muy amplio y correctamente etiquetado. Por lo que es inevitable tener que etiquetar todas y cada una de las imágenes. Aquí entran en acción los algoritmos de visión artificial.

En este apartado se pretende desarrollar varios algoritmos de procesamiento de imágenes para la creación del *"ground truth"* de manera automática que posteriormente se utilizará para entrenar la red neuronal.

En este caso para la elaboración de las etiquetas del *"ground truth"* se codificará la profundidad de las imágenes con el fin de poder identificar la áreas más cercanas y por lo tanto las susceptibles de ser esquivadas.

Una vez se han llevado a cabo varios entrenamientos, se utilizará RMSE como valor indicador para observar qué algoritmo consigue que la red neuronal aprenda con más facilidad. El principio de funcionamiento de estos códigos se basará en identificar los objetos lo suficientemente cercanos como para representar peligro de colisión.

4.8.2. Imágen estereoscópica

En la conducción autónoma todos los elementos a ser esquivados tienen un parámetro en común, la proximidad. Por ello la extracción de la profundidad es clave para poder desarrollar este tipo de algoritmos.

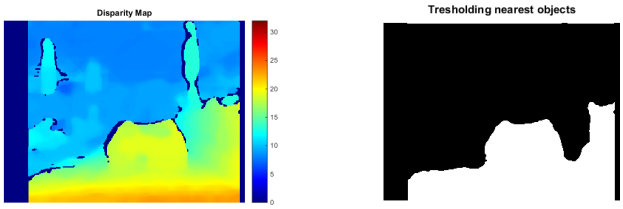
Las imágenes esterescópicas están formadas por dos imágenes tomadas desde ángulos ligeramente diferentes que permitirá, mediante trigonometría calcular la distancia de los objetos. Surgío como una analogía al funcionamiento de la vista en los seres humanos. La distancia entre los dos ojos permite al cerebro ser capaz de procesar e identificar que objetos están más o menos lejos.

Un procesador recibe dos imágenes digitales que, para él no son más que una agrupación de números sin sentido aparente, por lo que aplicar los principios de trigonometría no es posible. La manera en la que opera, es más sencilla. Ambas imágenes representan la fotografía de un objeto desde diferentes ángulos, por lo que si el objeto es rojo ambas imágenes tendrán píxeles con el mismo código de color. Pues bién, lo que hace el procesador es identificar los píxeles con la misma codificación de color y calcular la disparidad entre ambas imágenes. De esta manera, una disparidad pequeña significará que el objeto esta cerca, mientras que una disparidad grande significará que está lejos y así píxel a píxel se consigue obtener el mapa de profundidad. A continuación se muestra un ejemplo.

Figura 15: Stereo single images

Estas dos imágenes se han tomado desde diferentes ángulos, se puede observar que ambas fotografían el mismo paisaje. (Carretera Trento, Italia) Para procesarlas habrá que escoger el número de niveles de disparidad, que viene a ser el número de rango de distancias que será capaz de identificar. En este caso se escogerá 32 (simpre debe ser múltiplo de 16). Procesándola para calcular el mapa de disparidad se obtiene la siguiente imágen 16a. Puesto que el nivel de disparidad es 32, aparecerá la profundidad codifiada en 32 colores. Por último se establece un umbral en el que se codifican los píxeles más próximos con 255 y los más lejanos con un 0, de manera que se obtiene una imágen booleana que indica las zonas posibles de colisión. 16b

Una vez conocidas las zonas posibles de colisión se tratará de crear los códigos que generen un único punto por donde el dron pueda circular esquivando las áreas de colisión.



(a) Mapa de disparidad (b) Binarización

Figura 16: Imágenes stereo procesadas

4.8.3. Desarrollo de algoritmos de esquivación

Una vez computada la profundidad como se ha explicado en apartados anteriores, se disponen de una imagen plana de un sólo canal con todos los píxeles codificados de 0 a 255, siendo 255 profundidad 0 metros, y 0 profundidad 100 metros.

Con esta información es muy sencillo establecer una proporción para obtener un radio de sensibilidad. Es decir establecer un umbral que convierta la imagen en binaria con aquellos objetos dentro del radio de sensibilidad y aquellos fuera. Se supone que el algoritmo entrará en funcionamiento siempre a una velocidad constante determinada. De acuerdo con la publicación [4] se escogerá la velocidad de equilibrio del ala fija voladora que son 12 m/s.
Llegado a este punto es necesario analizar la dinámica del UAV para saber la magnitud que manejará entre tiempo y distancia para saber cuanto tiempo necesitará para hacer el desplazamiento requerido para esquivar el objeto.

En la imagen 17 se muestra la dinámica longitudinal del UAV tras implementar los controladores desarrollados en la publicación [4]. Se puede observar que se estabiliza a los 10 segundos tras demandar una variación de altura de 10 metros. Cabe destacar, que esta dinámica no es la única y faltaría por analizar la dinámica lateral. Aún así sirve para hacerse una idea de la magnitud.
Se concluye que es capaz de desplazarse un metro cada segundo.

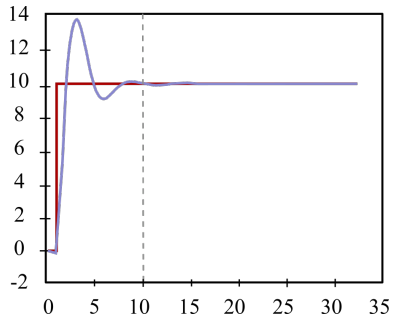


Figura 17: Respuesta longitudinal a una perturbación en altura

Volando a 12 m/s, cada segundo que pasa el dron avanzará 12 metros y será capaz de desplazarse longitudinal y/o lateralmente 1 metro. Por lo tanto si se quiere esquivar objetos en un rango de 6 metros lateralmente, El dron deberá recibir la información con 6 segundos de antelación, lo que se traduce en 72 metros y por lo tanto un **valor de offset entre 60 y 70.**

Posicionamiento de las áreas de colisión

Cada píxel de todos y cada uno de los fotogramas, tiene una posición referenciada en coordenadas cartesianas [x,y], con origen en la esquina superior izquierda. **Los algoritmos que se describirán a continuación tratarán de identificar donde están los píxeles de colisión (Píxeles blancos) para calcular un punto objetivo que garantice una trayectoria segura para el UAV.** Para ello, se ha desarrollado la siguiente función capaz de identificar la posición y el número de píxeles de colisión común a todos los algoritmos explicados a continuación:

```
def find_center(stereo):


(xmax,ymax)=stereo.shape
for i in range (0,xmax):
    for n in range (0,ymax):
       if stereo[i,n]> 60: # Transformacion de la imagen en binaria
         stereo[i,n]=255 # Cambia el valor del pixel a blanco puro
      else:
         stereo[i,n]=0 # Cambia el valor del pixel a negro puro
         points.append((i,n)) # Guarda solo los puntos negros
         posx=posx+i
         posy=posy+n
         count+=1


posx=posx/count
posy=posy/count
return (posx,posy,points)
```

En este proyecto se han ido probando diferentes alternativas a modo de investigación, a continuación la explicación:

Cálculo de centro de masas

Este algoritmo es el más simple de todos. Trata de buscar un punto equivalente que refleje la posición de todos los píxeles que no sean de colisión. Este punto sería el centro de masas del conjunto de puntos y sería calculado según la fórmula del centro de masas.

$$x_c = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i}; \qquad y_c = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i}$$

$$P_{obj} = [x_c, y_c]$$

Este método es muy eficiente y tiene un costo computacional relativamente bajo comparado con los demás. El problema surge cuando se dan morfologías singulares que inducen a error. Las más simples de entender son las mostradas en la siguiente imagen, en la que el centro de masas coincide en los píxeles de colisión, con lo cual se estaría guiando al UAV directamente hacia el obstáculo. En el segundo caso, el centro de masas coincide en un área de no colisión, pero no lo suficientemente segura.

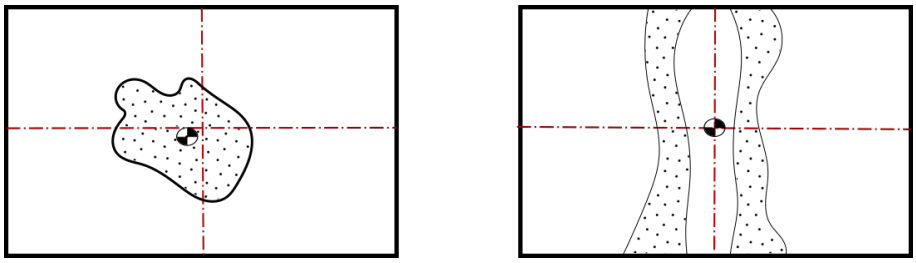


Figura 18: Singularidades del algoritmo de centro de masas

Cálculo de líneas de regresión

Este algoritmo trata de solucionar las singularidades anteriores. El principio de funcionamiento es muy similar pero esta vez se tratarán de ubicar los puntos de colisión mediante una recta de regresión. De esta manera, se tienen más certezas de que el punto de regresión, en el caso de la primera singularidad, no será ubicado en la zonas de colisión.

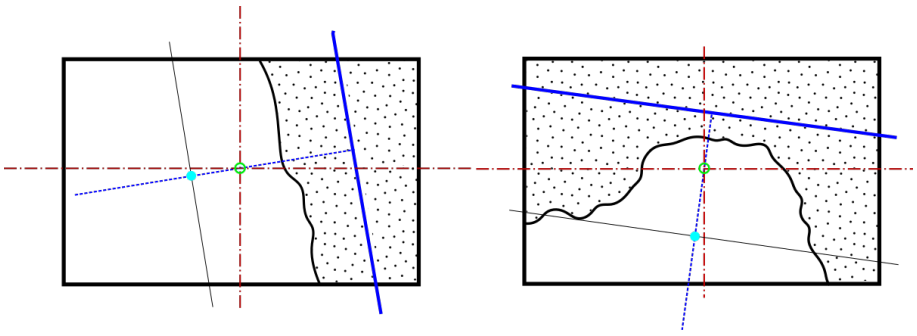


Figura 19: Ejemplo de aplicación del algoritmo de regresión

Con la recta de regresión ubicada y definida por 6, se trazará una perpendicular que pase por el punto central de la imagen [200,200]. (el cual indica la posición natural del UAV). Esta recta estará definida por un punto y su pendiente, la cual será la inversa de la de regresión. 7

El punto objetivo se ubicará en algún punto de la recta perpendicular, una vez pasado el punto central de la imagen. La distancia entre el punto objetivo y el centro, será inversamente proporcional a el número de pixeles de colisión respecto a el total de píxeles de la imagen 8. Esto es así porque si la línea de regresión está muy alejada del centro, significa que hay pocos puntos de colisión y por lo tanto la desviación necesaria para superarlo es muy pequeña. Por lo contrario, si la línea de regresión está muy cerca del punto central, significará que hay muchos puntos cerca de la futura posición del UAV, por lo tanto el esfuerzo para esquivarlo deberá ser mayor. Con este algoritmo se salvan las singularidades anteriores, pero surgen

algunas nuevas que, a pesar de ser solventables tienen un costo computacional muy grande.

$$Ecuacion\,punto\,pendiente : (y_r - y_1) = m(x_r + x_1) \tag{6}$$

$$Rectas\,perpendiculares\ : m_1 \cdot m_2 = -1 \rightarrow m_{perpendicular} = \frac{-1}{m_r} \tag{7}$$

$$d = \frac{200 \cdot (p(255))}{h \cdot l} \tag{8}$$

En python,la manera más sencilla de definir todas las rectas necesarias es encontrar la intersección de estas con los bordes de las imágenes. En el código se traduce de la siguiente manera

```
[vx,vy,x,y] = cv2.fitLine(points,cv2.DIST_L2,0,0.01,0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((stereo.shape[1]-x)*vy/vx)+y)
line1=[(righty,400),(lefty,0)]
nx,ny=1,-vx/vy
mag = np.sqrt((1+ny**2))
vx2,vy2 = nx/mag,ny/mag
```

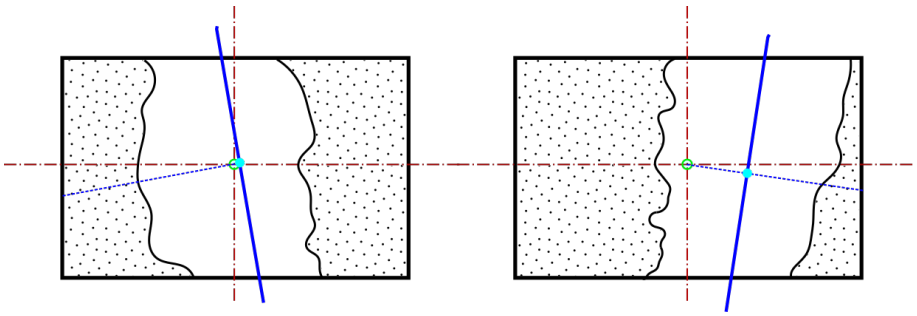


Figura 20: Singularidades del algoritmo de regresión

En este caso, habrá que analizar por código si la recta de regresión está o no sobre puntos de colisión. En el caso de no estar sobre puntos de colisión se ubicará el punto objetivo en la intersección de las dos rectas.

Otro problema a solucionar es que normalmente los objetos a esquivar se encuentran la parte mitad inferior de la imagen por el hecho de que normalmente estos están apoyados en el suelo. Lo cual supone, que la red neuronal sólo aprenderá a esquivar este tipo de obstáculos y de aparecer un en sustentación no sabría como reaccionar puesto que no ha sido entrenada para ello. Este problema se trata de solucionar volteando 180 grados la mitad de imágenes destinadas a entrenamiento, lo cual desemboca en otra dificultad.

A la hora de calcular la distancia entre el punto de objetivo y el punto central de la imagen existen dos posibles soluciones (Una a cada lado del punto central) y sólo una de ellas es correcta. Cuando los obstáculos están en la parte inferior de la pantalla se utilizará una solución, mientras que cunado se procesen imágenes volteadas, habrá que utilizar la otra.
En código se trata de la siguiente manera:

```
def calcular _ distancia (d,m,n,x2,y2,sol):
    x1=0
    y1=0
    y1 = Symbol('y1')
    try:
       a,b=solve( ((200-(m*y1+n))**2+(200-(y1))**2)-d**2 ,y1)
       if sol==True:
          sb=a
       else:
          sb=b
return (sb)
```

Este algoritmo ha sido probado en vídeos estéreo reales produciendo muy buenos resultados. En la figura de a continuación se puede observar como el algoritmo sitúa el punto objetivo para esquivar un vehículo.
No obstante el coste computacional es bastante alto, y la gestión y cálculo de rectas produce muchos errores en ejecución.

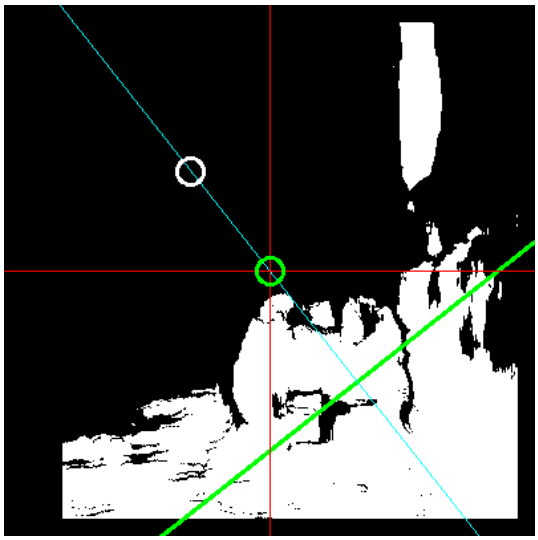


Figura 21: Ejemplo de aplicación a un caso real

Sectorización y media de subimágenes

Este código trata de eliminar todas las singularidades vistas anteriormente, integrando la facilidad de cálculo del algoritmo de centro de masas.
Trata de dividir la imagen en cuatro subimágenes por cuadrantes y calcular el centro de masas de cada una de ellas. Paralelamente se cuenta el punto de píxeles negros (De no colisión) para poder identificar que cuadrantes son los menos peligrosos para dirigir el UAV. Una vez se tiene esta información, se ordena de mayor a menor los cuadrantes según la peligrosidad y se seleccionarán los tres con menos píxeles de colisión para continuar los cálculos.

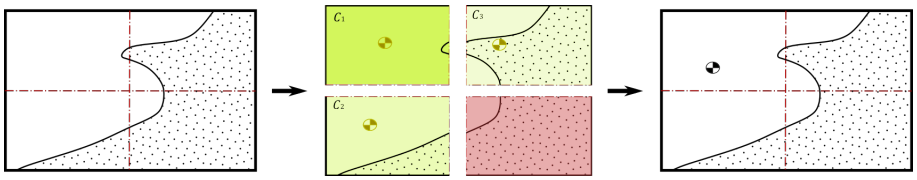


Figura 22: Algoritmo de sectorización

Para calcular el punto objetivo final, se volverá a utilizar la fórmula del centro de masas. Cabe destacar que la sectorización, permite ponderar la fórmula según la cantidad de puntos de colisión, es decir se puede dar más peso mediante un coeficiente a aquel cuadrante que presente menos píxeles de colisión. De esta manera la fórmula del punto objetivo quedará de la siguiente forma:

$$x_c = \frac{C_1 m_1 x_1 \cdot C_2 m_2 x_2 \cdot C_3 m_3 x_3}{\sum_{i=1}^{3} m_i} \tag{9}$$

$$y_c = \frac{C_1 m_1 y_1 \cdot C_2 m_2 y_2 \cdot C_3 m_3 y_3}{\sum_{i=1}^{3} m_i} P_{obj} = [x_c, y_c] \tag{10}$$

$$P_{obj} = [x_c, y_c] \tag{11}$$

Finalmente se muestra la función encargada de llevar todo este proceso acabo.

```
def pobj(stereo):

 sub_img1 = np.zeros((100,100))
 sub_img2 = np.zeros((100,100))
 sub_img3 = np.zeros((100,100))
 sub_img4 = np.zeros((100,100))

 sub_img1 = stereo[0:100,0:100]
 sub_img2 = stereo[100:200,0:100]
 sub_img3 = stereo[0:100,100:200]
 sub_img4 = stereo[100:200,100:200]

 p1,p2,data1,sub_img1=find_center(sub_img1)
 p3,p4,data2,sub_img2=find_center(sub_img2)
 p5,p6,data3,sub_img3=find_center(sub_img3)
 p7,p8,data4,sub_img4=find_center(sub_img4)

 coll1=float(data1.shape[0])/10000
 coll2=float(data2.shape[0])/10000
 coll3=float(data3.shape[0])/10000
 coll4=float(data4.shape[0])/10000

 datalist=[[data1,(p1,p2),coll1],[data2,(p3+100,p4),coll2],
      [data3,(p5,p6+100),coll3],[data4,(p7+100,p8+100),coll4]]
 datalist.sort(key=lambda x: x[0].shape,reverse = True)

 c1=datalist[0][2]
 c2=datalist[1][2]
 c3=datalist[2][2]
 c4=datalist[3][2]

 x1=datalist[0][1][0]
 x2=datalist[1][1][0]
 x3=datalist[2][1][0]

 y1=datalist[0][1][1]
 y2=datalist[1][1][1]
 y3=datalist[2][1][1]

 m1=datalist[0][0].shape[0]
 m2=datalist[1][0].shape[0]
 m3=datalist[2][0].shape[0]

 imgfin = np.zeros((200,200))

 p1,p2=datalist[0][1]
 p3,p4=datalist[1][1]
 p5,p6=datalist[2][1]

 c1=c1*1.5

 if (x1 and y1 and y2 and x3)==50:
  X=100
  Y=100
 else:
  X=(x1*m1*c1+x2*m2*c2+x3*m3*c3)/(m1+m2+m3)
  Y=(y1*m1*c1+y2*m2*c2+y3*m3*c3)/(m1+m2+m3)

 return(int(X),int(Y),imgfin,p1,p2,p3,p4,p5,p6)
```

4.9. Entrenamiento de la red neuronal

4.9.1. Introducción

En el entrenamiento la Red Neuronal aprende cómo comportarse ante determinadas entradas. Esto lo hace asignando determinadas ponderaciones a las salidas de las etapas intermedias que irán cambiando con el fin de minimizar el error con respecto a el *"ground truth"*, que es la salida teórica de la RN.

En este caso los datos de entrenamiento serán imágenes con la profundidad codificada, y el *"ground truth"* sería los puntos objetivos generados a través de los algoritmo explicados anteriormente.

De esta manera, se introduce la imagen de entrenamiento a la red neuronal que arrojará una salida. Esta será comparada con el *"ground truth"* y según sea más o menos correcta variarán los pesos o ponderaciones de las etapas intermedias de la RN.

Tras terminar el entrenamiento se observa la función de error y se escogerá la etapa que mejor características presente. Las ponderaciones de esa etapa será las utilizadas en la aplicación real.

4.9.2. Selección del número de etapas

Entrenar una red neuronal requiere un proceso de optimización continuo para minimizar el error lo máximo posible. Esto es difícilmente conseguible a la primera puesto que hay bastantes parámetros que se pueden variar. Por ello, es necesario correr varios entrenamientos e ir observando errores e ineficiencias para corregirlas.

Dadas las limitaciones de la máquina virtual creada no se pueden ejecutar tantos experimentos como sería recomendable (Cada entrenamiento con una media de mil imágenes tarda unas 10 horas en ejecutarse).
Será importante establecer un número de etapas que optimice el tiempo de procesamiento y el aprendizaje de la red neuronal. Para ello, se ha ejecutado un primer entrenamiento de todas de ellas para poder observar la evolución de la función de pérdidas (*"Loss function"*)

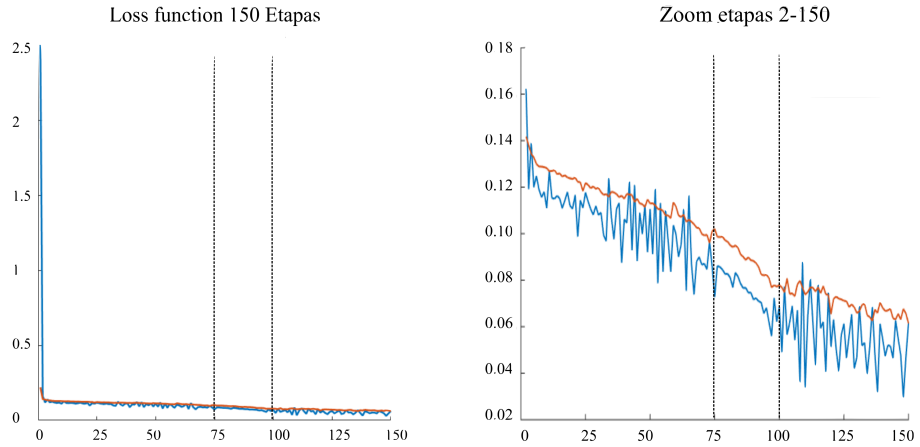


Figura 23: Evolución de las pérdidas en 150 etapas

Se pueden observar 3 fases en el entrenamiento. Se han delimitado entre las dos líneas discontinuas.
En la primera parte, hay grandes oscilaciones en los valores debido a la aleatoriedad inicial de los pesos, se podría decir que es un estado transitorio. En la segunda, las variaciones se estabilizan y se observa una disminución del error continua hasta la

tercera etapa donde vuelven a aparecer oscilaciones. Estas son el primer síntoma del llamado *"Overfitting"*, se debe a que los pesos empiezan a ser demasiado precisos y pierden flexibilidad ante nuevos escenarios. Funcionará bien cuando la imagen a aprender sea visiblemente parecida a alguna ya aprendida, y arrojará resultados erróneos al recibir una imagen a aprender no parecida a ninguna anterior. Esto se quiere evitar a toda costa por lo tanto se deberá reducir el número de etapas.

A continuación se estudiará la variación del error a lo largo de diferentes etapas.

Training
$$\begin{cases} E(10) - E(150) = 0.1980 - 0.06154 = 0.13646 \\ E(50) - E(150) = 0.1106 - 0.06154 = 0.04906 \\ E(100) - E(150) = 0.1152 - 0.06154 = 0.03728 \end{cases}$$

De esta manera se puede observar, que la mayor reducción de error esta ocurriendo en las primeras etapas de entrenamiento. De esta manera se considera conveniente que **para los siguientes entrenamientos se limite el número de etapas a 100.**

4.9.3. Selección del algoritmo óptimo

Para la sección del algoritmo óptimo de una manera rápida sencilla y visual, se graficarán todas los resultados predichos junto a los reales, de manera que se pueda observar a simple vista cómo se comporta la red neuronal por si sola. Estos gráficos son llamados los histogramas y también son generados mediante código. Se graficará la salida de ambas componentes tanto la X como la Y.

Como indicadores numéricos se utilizarán el RMSE (*Root Mean Square Error*). Es una medida de error muy común en el mundo de la estadística. Permite conocer la medida del error en valor absoluto. Se corresponde con la raiz cuadrada de los "errores" de cada valor al cuadrado. El error en cada punto es llamado residuo.

$$RMSE = \sqrt{(X_{True} - X_{pred})^2} \tag{12}$$

4.9.4. Histogramas

Se utilizarán para contrastar de manera sencilla los resultados de la red neuronal con las etiquetas reales. Estos se mostrarán agrupados según el valor de la salida.

Cabe destacar que dado que el RMSE es un coeficiente que incorpora un sumatorio se decide dividir el valor por el número de muestras para relativizarlo en caso de que el número de muestras no sea el mismo en los diferentes experimentos. Por lo tanto la fórmula queda de la siguiente forma:

$$RMSE_m = \frac{\sqrt{\sum (prediccion - real)^2}}{N_{muestras}}$$

Centro de masas

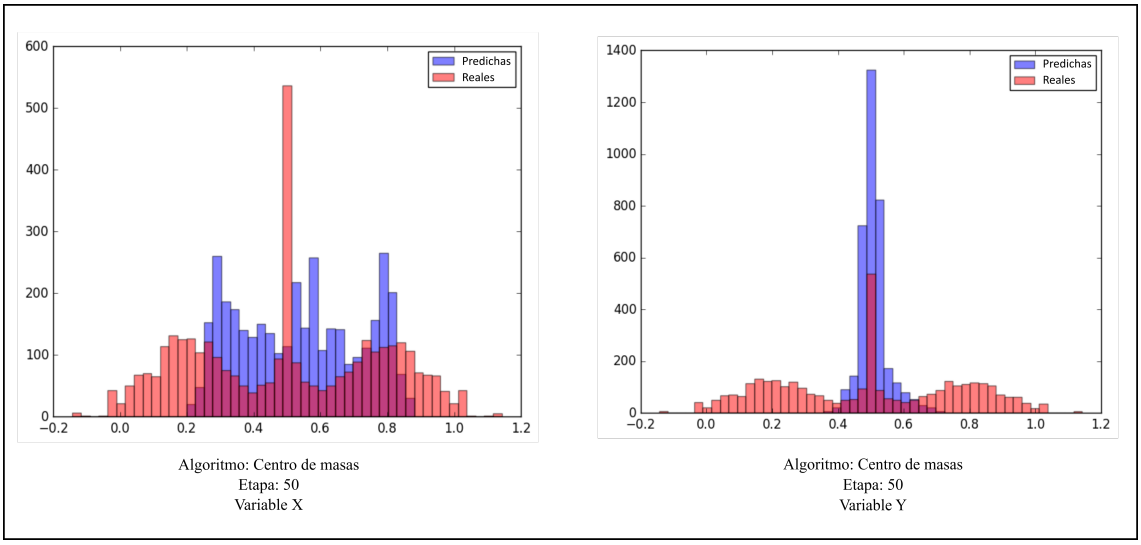


Figura 24: Resultados teóricos y reales en el algoritmo: Centro de Masas

RMSE(x): 0.543

RMSE(y): 1.0030

En este caso la red neuronal no es capaz de extraer la información de las imágenes y se comporta de manera inconsistente tal y como se puede observar en el histograma. Este método es desestimado.

Regresión

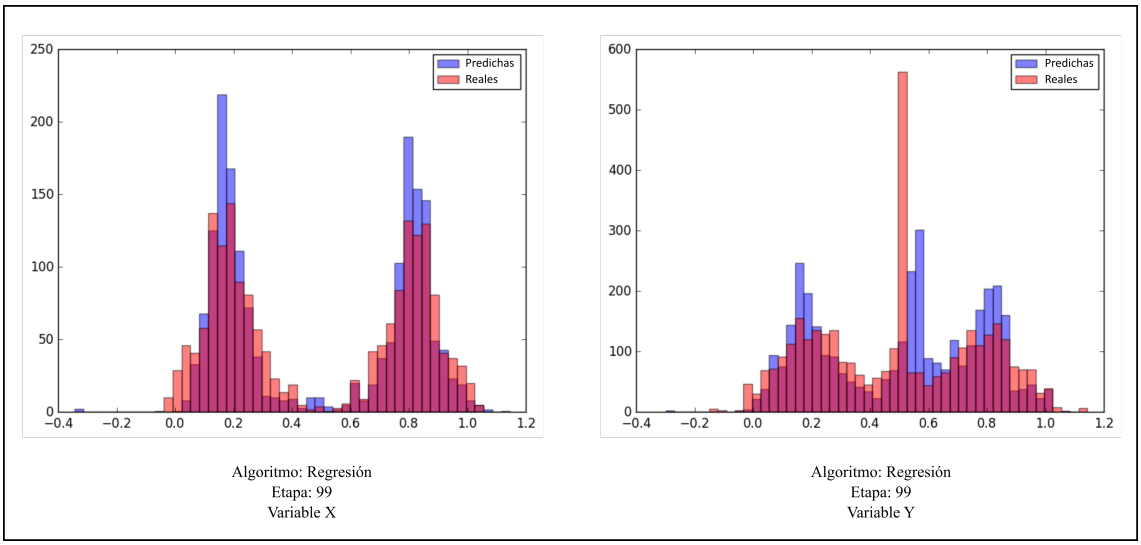


Figura 25: Resultados teóricos y reales en el algoritmo: Regresión

RMSE(x): 0.0048

RMSE(y): 0.0030

En este caso, se puede observar un comportamiento mucho mejor. El valor del RMSE es mucho menor y comparado con el resto de algoritmos es el que mejor comportamiento presenta.

Sectorización

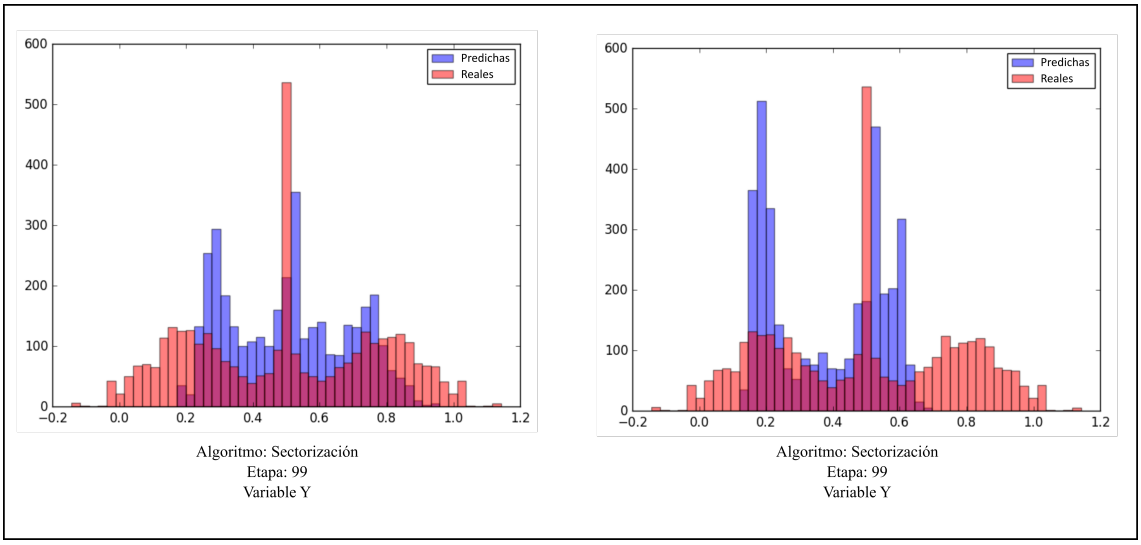


Figura 26: Resultados teóricos y reales en el algoritmo: Sectorización

RMSE(x): 0.6509

RMSE(y): 0.9672

Finalmente la red neuronal tampoco es capaz de aprender correctamente con este algoritmo, por lo que se desestima.

4.9.5. Análisis de error y selección de la etapa óptima

Con el algoritmo ya seleccionado se realiza un estudio más profundo para evaluar como ha sido el aprendizaje. En la imagen de a continuación se muestran las gráficas arrojadas tras los entrenamientos de ambas coordenadas.

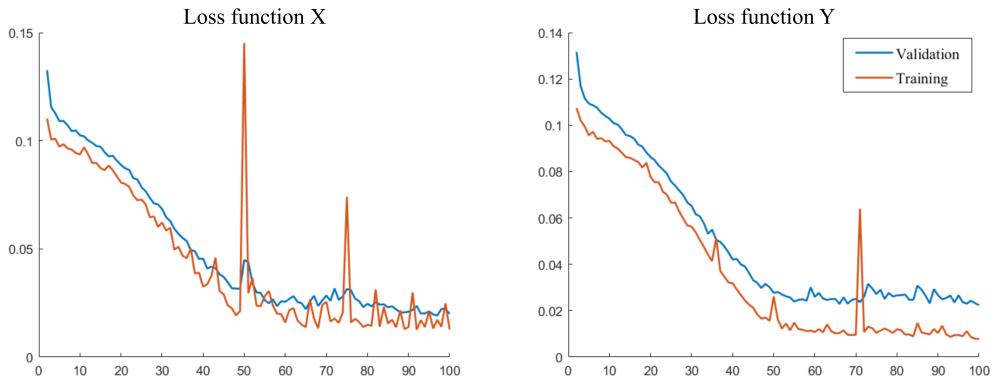


Figura 27: Loss function

En la coordenada "X", se puede observar que tras la reducción de error constante hasta las 50 etapas, aparecen picos aleatorios cuyo origen se ha explicado antes. Por ello para la aplicación se utilizarán los pesos de la etapa 49. El comportamiento en la variable "Y" es muy parecido aunque los picos son más moderados. Por ello también se utilizarán los pesos de la **etapa 49**.

4.10. Desarrollo del entorno en ROS

4.10.1. Introducción

En este proyecto, ROS permitirá realizar el procesado de las imágenes de una manera sencilla y ordenada con un mínimo coste computacional. A demás, dada la voluntad de crear un sistema con costo económico bajo, se convierte en una excelente opción al ser compatible con todo tipo de SO open source.

ROS, (Robot Operating System) es un sistema metaoperativo de código abierto enfocado a optimizar la eficiencia de sistemas robóticos permitiendo un desarrollo de software en un entorno seguro y sencillo de utilizar. Surgió de la necesidad de dar soporte al primer robot con inteligencia artificial (STAIR). Posteriormente se permitió su uso gratuito a todo tipo de usuarios con el fin de implantar un protocolo estándar en el mundo de la robótica.

Proporciona los servicios esperados de un sistema operativo, incluida la abstracción de hardware, el control de dispositivos de bajo nivel, la implementación de la funcionalidad de uso común, el paso de mensajes entre procesos y la gestión de paquetes. También proporciona herramientas y bibliotecas para obtener, crear, escribir y ejecutar código en varias computadoras de manera simultánea.

Para este proyecto es una manera sencilla de manejar toda la información y simular con riesgo cero todo el funcionamiento del proyecto.

4.10.2. Conceptos

Nodo. Los nodos son procesos que realizan cálculos y procesos. Están programados en Python o en C++ para los cuales hay librerías de soporte como roscpp y rospy en este proyecto se han programado y modificado varios nodos en ambos lenguajes . Es muy importante dedicar un tiempo a diseñar la arquitectura nodal y el intercambio de información entre nodos para conseguir la máxima eficiencia y no ralentizar el sistema debido a una sobrecarga de información entre nodos. Más adelante se mostrará el esquema nodal de este sistema. Dentro de los nodos, existen dos categorías:

- ***"Subscriber"***. Son aquellos nodos destinados a leer información de algunos de los *"Topics"*.
- ***"Publisher"***. Aquello nodos capaces de enviar información a través de los *"Topics"*.

En ambos caso, su programación es muy sencilla gracias a las bibliotecas específicas. A modo de ejemplo se muestra las líneas para suscribir un código a un tópico y publicar información en el mismo. Simplemente se ha de definir un objeto del tipo *Publisher* indicando el tópico al que se va a publicar. Para publicarlo únicamente hará falta utilizar el **método** *publish* con la variable que se quiera transferir.

```
pub = rospy.Publisher('chatter', String, queue_size=10)
rospy.init_node('talker', anonymous=True)
pub.publish(hello_str)
```

En este caso es mucho más sencillo simplemente habrá que indicar el nombré del tópico y el tipo de dato a leer.

```
rospy.init_node('listener', anonymous=True)
rospy.Subscriber('chatter', String, callback)
```

ROS Master. Es el nodo principal que permite la comunicación entre todos los nodos secundarios.

Topic. Dicho vulgarmente serían *"buses"* utilizados por los nodos para transmitir información. Son capaces de transmitir cualquier tipo de información, desde imágenes hasta simples integrandos. El tipo de dato que vaya a transmitir determinará el tipo de tópico a crear.

Generalmente los nodos no son conscientes de con que otros nodos se comunican, son estos los que tienen que suscribirse para poder acceder a los datos que les interesen. Puede haber varios *"Publishers"* y *"Talkers"*.

Catkin. Es el compilador oficial de ROS. Combina macros CMake y scripts de Python para lograr el menor costo computacional y mejorar el flujo de trabajo. Entre las ventajas de este compilador destaca su gran portabilidad y flexibilidad que permiten su uso en gran variedad de equipos y sistemas operativos, siempre basados en Linux.

Packages. El software esta organizado en paquetes. Estos pueden contener un conjunto de nodos, librerías, archivos de configuración ... etc. La principal funcionalidad de ellos es conseguir una fácil reutilización de software.

Son sencillos de crear a mano o con herramientas como *"catkin"*. Son la unidad más pequeña que compone ROS, básicamente son un directorio que tras compilarse contiene un archivo *package.xml* en él . Este archivo define las propiedades del paquete como su nombre, versión autores y dependencias.

4.10.3. Implementación y utilidad de ROS

Existen muchas versiones de ROS que han ido evolucionando desde su creación. En este caso, por temas de compatibilidad con la máquina virtual utilizada se ha escogido la versión *kinetic*.

En la siguiente imagen se muestra un esquema básico de como funciona el sistema diseñado.

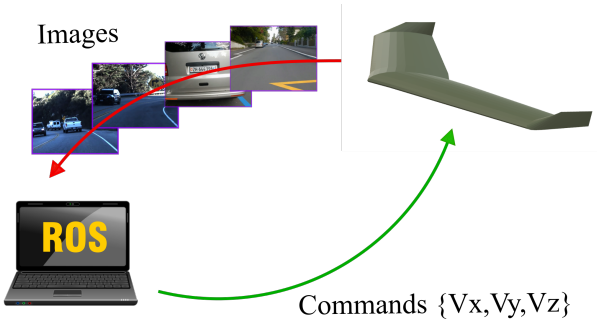


Figura 28: Marco de uso de ROS

El dron adquirirá las imágenes y las enviará mediante WiFi al ordenador, también cabe la posibilidad de que sean enviadas vía radio añadiendo un conversor analógico-digital a la entrada del ordenador. Habría que estudiar los tiempos de latencia, aspecto que no se aborda dadas las limitaciones de tiempo.

El ordenador, con un sistema operativo basado en Linux recibe las imágenes y las procesa en los diferentes nodos de ROS que se detallarán más adelante. El resultado de este procesamiento serán referencias en velocidad que serán enviadas mediante radio al UAV cerrando así el lazo.

Mediante el siguiente comando cmd, ROS muestra una ventana gráfica con un esquema que muestra la arquitectura nodal que esta utilizando en ese momento. En un óvalo representa los nodos y en un rectángulo los topics.

```
rosrun rqt_graph rqt_graph
```

Se ha ejecutado en una simulación que cubre desde la adquisición ficticia de imágenes (Puesto que la parte UAV-PC no está cubierta) hasta el envío de referencias por radio a el UAV.

El nodo 1, Nodo de simulación, es un nodo creado únicamente para publicar las imágenes en el topic "/UAV/image_raw" y que puedan ser utilizadas posteriormente por otro nodo.

Grupo de topics 2, simula los topics que generaría estar conectado de verdad a el UAV. Contiene dos topics, el anterior mencionado con las imágenes y el "/UAV/state_change" que indica el estado en el que se encuentra el UAV, por ejemplo, vuelo autónomo o manual.

El nodo 3, Nodo de percepción, procesa las imágenes en la red neuronal y extrae la información de las mismas, que posteriormente será publicada en el topic 4 "cnn_out/predictions".

El nodo 5 realiza la transformación de la información extraída de la red neuronal a referencias de control que serán enviadas por RC al UAV. Sera explicado en el apartado de control.

El topic 6 simplemente pasa la probabilidad de colisión del nodo 1 a el 5.

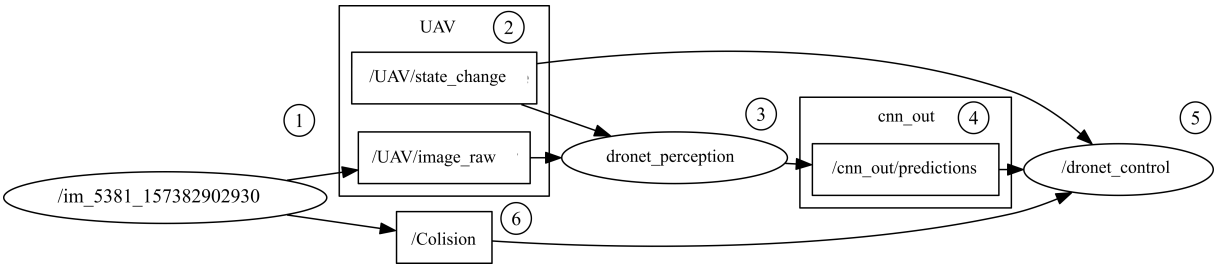


Figura 29: Nodos

4.10.4. Nodo de simulación

Para poder programar toda la arquitectura de manera sencilla es necesario poder simular el UAV, de lo contrario habría que estar estableciendo la conexión cada vez que se quisiese probar cualquier fragmento de código. Para evitar esto, se ha creado un nodo que publique las imágenes y variables tal y como entrarían en el lazo de control.

Se publicarán dos variables en el entorno de ROS. Una será la imágen y otra la probabilidad de colisión calculada. Para poder publicar ambos hay que inicializar el método *rospy.Publisher* con sus atributos correspondientes, que en este caso serán en que topic se va a publicar (Ej.:*'/bebop/image_raw'*), el tipo de dato que se va a publicar que tendrá que ser importado de la clase *std_msgs.msg* con antelación y por último la dimensión de la pila antes de sobrepasarse y perder datos. A conrinuación se muestran las dos necesarias para el nodo de simulación:

```
Imge=rospy.Publisher('/bebop/image_raw',Image,queue_size=10)
Coll=rospy.Publisher('/Colision',Float32,queue_size=10)
```

Otra peculiaridad de operar de este nodo es que al operar con ROS y la librería OpenCV, es necesario utilizar una función puente (CvBridge) que codifique la imagen para poder ser publicada en un topic. A continuación se muestra la llamada a la función y al método *publish* que permite publicar los mensajes.

```
msg_frame=CvBridge().cv2_to_imgmsg(stereo,encoding='mono8')
Imge.publish(msg_frame)
Coll.publish(colision)
```

Por último la función que calcula la probabilidad de colisión. Para ello será necesario conocer las dimensiones de la imagen para calcular el número total de píxeles y así poder calcular el porcentaje.

```
def Colision(stereo):


      (xmax,ymax)=stereo.shape
      colision = np.sum(stereo > Threshold)
      colision=count/(xmax*ymax)


return (colision)
```

De esta manera se da solución a la problemática presentada incluyendo la entrada de imágenes y el bloque de preprocesamiento.

4.10.5. Nodo de percepción (CNN)

Este nodo contendrá la red neuronal. Estará suscrito al topic de las imágenes (*/UAV/image_raw*) y publicará las coordenadas X,Y resultado (*/cnn_out/predictions*). Este código cargará la estructura de la red neuronal del archivo *.json* generado durante el entrenamiento de la red neuronal. Los pesos tendrán que ser ubicados en el directorio correspondiente para que puedan ser reconocidos por el script.

A continuación se muestra la función principal.

```
class CNN(object):
   def __init__(self,
            json_model_path_X,json_model_path_Y,
            weights_path_X,weights_path_Y, target_size=(200, 200),
            crop_size=(150, 150),
            imgs_rootpath="../models"):

      self.pub = rospy.Publisher("cnn_predictions", CNN_out, queue_size=5)
      self.feedthrough_sub = rospy.Subscriber("state_change", Bool,
              self.callback_feedthrough, queue_size=1)
      self.land_sub = rospy.Subscriber("land", Empty, self.callback_land, queue_size=1)

      self.use_network_out = False
      self.imgs_rootpath = imgs_rootpath

      K.set_learning_phase(TEST_PHASE)

      model1 = utils.jsonToModel(json_model_path_X)
      model2 = utils.jsonToModel(json_model_path_Y)

      model1.load_weights(weights_path_X)
      model2.load_weights(weights_path_Y)
      print("Loaded model from {}".format(weights_path_X))

      model1.compile(loss='mse', optimizer='sgd')
      model2.compile(loss='mse', optimizer='sgd')

      self.model1 = model1
      self.model2 = model2

      self.target_size = target_size

      self.crop_size = crop_size
 print 'Process start'
```

4.11. Aplicación al guiado autónomo de un UAV

4.11.1. Introducción

Todo lo explicado anteriormente está enfocado a la extracción de información del entorno, no obstante no es capaz de generar referencias de control para esquivar cualquier tipo de objeto. En este apartado, se explicará la estrategia de control final, que a su vez está implementada dentro de ROS.

Únicamente con la información extraída de la red neuronal, no es posible controlar el dron. Será necesaria idear una estructura de control que transforme las referencias generadas a través de la red neuronal, en señales de control comprensibles para el UAV.

4.11.2. Arquitectura de control

Se presenta una arquitectura en cascada con un lazo externo que contendrá la red neuronal. Esta generará referencias de velocidad lineal para el lazo interno que junto a las variables dinámicas y las variables de posición del UAV extraídas de la IMU, asegurará la estabilidad y consecución de las referencias.
La arquitectura ideada es la siguiente:

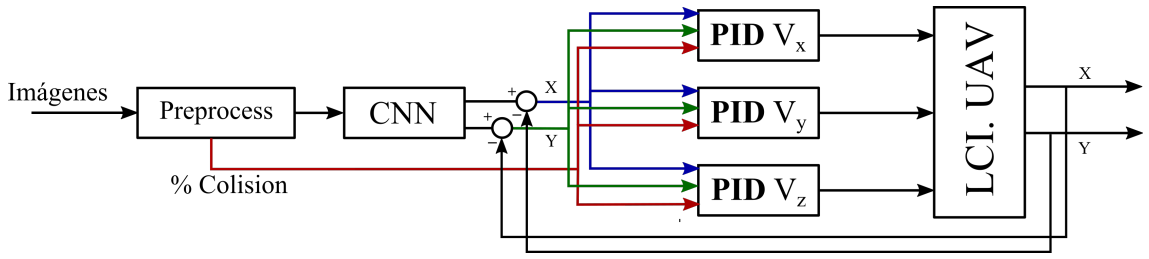


Figura 30: Arquitectura de Control

La entrada son las imágenes capturadas con el sistema diseñado para este proyecto explicado en otro apartado.

En la caja **Preprocess** se contarán los puntos de colisión que tal y como se ha explicado anteriormente, serán aquellos cuyo valor sea mayor a un nivel previamente establecido dependiendo de las características de la composición de cámaras. Este código es muy sencillo y eficiente puesto que las librerías de Python de tratamiento de imágenes están muy enfocadas a llevar a cabo este tipo de operaciones. La salida de esta caja será la probabilidad de colisión y la imágen intacta como ha sido introducida.

En el siguiente paso las imágenes se introducen en la **CNN** de donde se extrae el punto objetivo que ayudará a esquivar el objeto.

Con las tres variables generadas hasta ahora $(X, Y, \%Colision)$ Se pueden diseñar los **PIDs** que generarán la referencia en velocidad lineal. Esta referencia irá al lazo de control interno (**LCI. UAV**), donde ya con los controladores propios del dron se conseguirá esquivar el obstáculo.

4.11.3. Control a partir de imágenes

En la aplicación real, será necesario procesar un flujo continuo de imágenes y generar las referencias de control a partir de ellas. El único dato constante será el número de fotogramas por segundo. Idealmente, rondará los 30 fps, aunque en este proyecto con el hardware lowcost ronda los 10 fps. No obstante los cálculos se realizarán suponiendo 30 fps por simplicidad para acercarse a una posible aplicación real.

Las imágenes recibidas, todas tendrán un punto objetivo que el dron deberá alcanzar. En la imagen siguiente se muestra como a partir de dos fotogramas, cada uno con su punto objetivo, se puede obtener información para poder controlar el sistema. 31

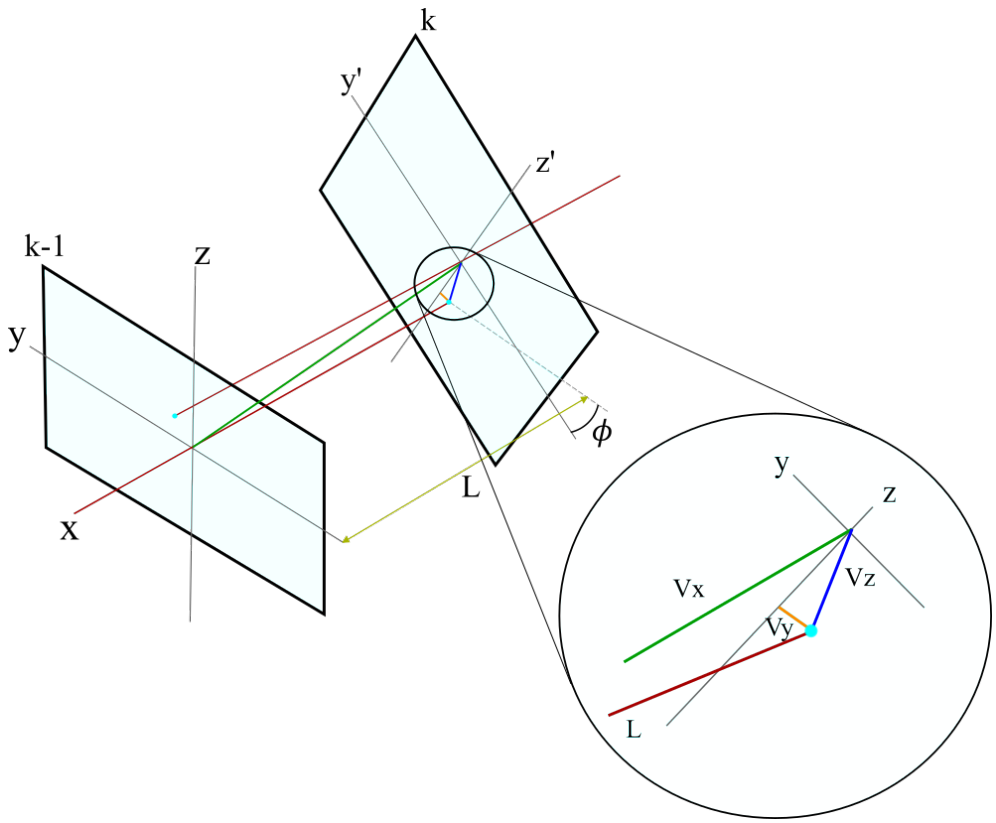


Figura 31: Conceptual image relating time and speed

En esta imagen aparecen los fotogramas “k-1” y “k”, tomadas en dos momentos consecutivos mientras el dron estaba en movimiento. En el fotograma “k” se supone que el dron ha conseguido alcanzar el punto objetivo del fotograma “k-1”. Para conseguirlo habrá tenido que generar una velocidad específica en los ejes “y” y “z”. Estas velocidades se obtendrán tras calcular la distancia de la proyección del punto objetivo sobre el punto central de la imagen anterior tal y como se muestra en la ampliación de la imagen.

Hay que resaltar que de cara a un futuro se puede implementar un control más robusto considerando aceleraciones, velocidades e inercias como entrada a la red neuronal. Esto supondría un gran costo ya que la generación del *“ground truth”* no sería automática y debería realizarse mediante experimentos en vuelo.

4.11.4. Implementación de los PIDs

Introducción al código

Se creará un nodo que tenga la función exclusiva de generar las referencias de control. Este nodo está programado en C++ ya que es más eficiente a la hora de ejecutar saltos entre funciones, muy necesario en este código en el que cada vez que haya un cambio en un topic saltará a su *Callback* correspondiente.

Los *Callback* son las funciones a donde salta en caso de que se de una condición definida con anterioridad. En este caso todas las condiciones serán que haya cambios en un topic.

De este nodo se reutilizará la clase definida *deepNavigation* por la ETH Suiza. En ella se procesan los datos desde la salida de la red neuronal a la generación de referencias de control.

Se añadirá un proceso llamado *CollisionCallback* donde se leerá el topic de colisión. Por último se añadirán todas las variables necesarias para llevar a cabo el algoritmo de control.

Descripción del código

En primer lugar habrá que inicializar la suscripción a los tópicos correspondientes (*cnn_out/predictions*, $\%Colision$)

```
network =nh_.subscribe("cnn_predictions", 1,
      &deepNavigation::deepNetworkCallback, this);
Coll_sub_ =nh_.subscribe("/Colision", 1,
       &deepNavigation::CollisionCallback, this);
```

Una vez se han realizado las subscripciones, cada vez que se registre un cambio en el topico se saltará automáticamente al Callback indicado en la subscripción. En este caso se muestra el Callback de la colisión. En el se guardan hasta dos muestras antiguas para utilizarlas más adelante.

```
void deepNavigation::CollisionCallback(const std_msgs::Float32& msg)
{
      Coll2=Coll1;
      Coll1=Coll;
      Coll = msg.data;
}
```

Por último se han implementado los tres controladores PID para las velocidades lineales. No han sido diseñados siguiendo estrategias de control puesto que no entra dentro de los objetivos del TFM. Su único objetivo es la validación de la capacidad de tratar la información para generar las referencias.

Generación de referencia en velocidad lineal V_x

Es la velocidad más importante ya que va ligada directamente a la cantidad de metros recorridos entre fotogramas, por lo tanto influye en el tiempo que será necesario para esquivar el objeto y en las velocidades V_y, V_z. También debe estar relacionada con la probabilidad de colisión. De esta manera cuanta más probabilidad de colisión haya, menor será la velocidad del eje X y menores serán también las velocidades en el eje Y y Z.

Se considera que la velocidad debe ser 0 m/s si la probabilidad de colisión es 0.9 o superior y que a partir de una probabilidad de 0.2 se puede alcanzar una velocidad de 15 m/s, que se corresponde con la velocidad de eficiencia óptima calculada en el TFG. Los rangos de operación establecidos son algo totalmente arbitrario que se han escogido siguiendo un razonamiento coherente, con el fin de poder seguir avanzando en el desarrollo de proyecto. Dados esos dos puntos se dibuja la recta mostrada a continuación cuya ecuación es:

$$y = -21.43 * x + 19.29$$

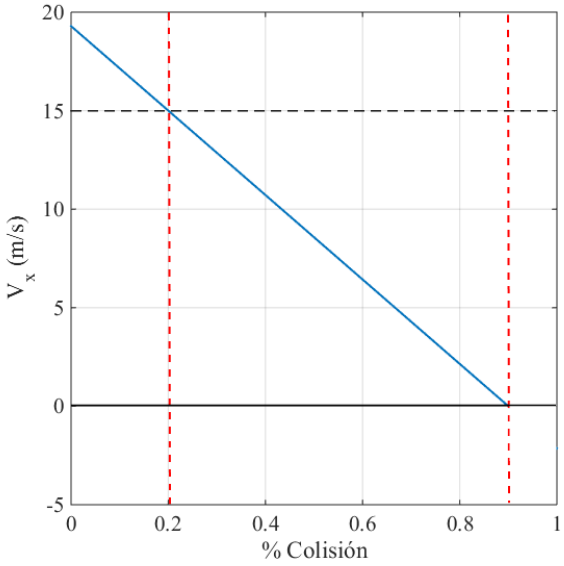


Figura 32: Recta Velocidad- %Colisión

Generación de referencia en velocidad lineal V_y e V_z

Ambas se corresponden con dinámicas laterales y los controladores se han diseñado de manera estimada, puesto que no es el objetivo principal de este proyecto. Se han implementado 2 PID's tipo para mostrar como sería el funcionamiento.

Finalmente se muestra el fragmento de código en el que se han implementado los controladores. Esta parte del código se ha programado en C++.

```
void deepNavigation::run()
{

  ros::Duration(2.0).sleep();
  ros::Publisher velocity_params = nh_.advertise
        < geometry_msgs::Twist > ("/hola", 10);
  ros::Publisher desired_velocity_pub_ = nh_.advertise
        < geometry_msgs::Twist > ("velocity", 5);
  ros::Rate rate(30.0);

  while (ros::ok())
  {

    Coll_=0.25*Coll1+0.1*Coll2+0.1*(Coll-Coll1)+Coll;
    Vx=-21.43*Coll_+19.29;  //m/s
    Vy=(0.1*(X2)+0.25*(X1)+(X)+((X)-(X1)));
    Vz=(0.1*(Y2)+0.25*(Y1)+(Y)+((Y)-(Y1)));
    // Publish desired
    if (use_network_out_)
    {
        desired_velocity_pub_.publish(cmd_velocity_);
    }
    else
 //velocity_params.publish(cmd_velocity_);
        ROS_INFO("NOT PUBLISHING VELOCITY");

    ROS_INFO("Vx: %.3f - Vy: %.3f - Vz: %.3f", Vx, Vy,Vz);
    ROS_INFO("--------------------------------------");


    rate.sleep();
    ros::spinOnce();

  }

}
```

4.12. Implementación de Hardware

4.12.1. Introducción

Actualmente las cámaras estereoscópicas tienen un coste muy elevado y presentan muy poca flexibilidad ya que los modelos comerciales son compactos con distancias entre cámaras fijas.

Para proyectos de este tipo interesa que el producto se adapte a las necesidades de la aplicación dentro de una capacidad presupuestaria reducida.

Por lo tanto, se decide diseñar el hardware basado en tecnología de bajo coste de cara a observar su comportamiento y la viabilidad en comparación con otros modelos comerciales.

El prototipo diseñado consta de una raspberry Pi 3 B+ que almacenará todas las imágenes y coordinará las comunicaciones con las otras dos Raspberries Pi Zero. Estas serán utilizadas a modo de driver para cada una de las cámaras.

Cabe destacar que en este proyecto la única funcionalidad que cubre este hardware es la adquisición de imágenes y **no** realiza en ella ningún tipo de procesamiento más halla de la calibración.

A continuación se explicará la configuración y el funcionamiento.

4.12.2. Conexión OTG Raspberry Pi zeros to Raspberry Pi 3

Se pretende conectar las dos raspberrys Pi Zero a la raspberry Pi 3 principal mediante USB, La RPi principal, deberá alimentar a las otras dos y acceder a ellas mediante SSH para poder transferir datos en las dos direcciones.

Dentro de las especificaciones a tener en cuenta, se utilizarán cables USB de transferencia datos y no de energía. Será necesario alimentar la RPi principapal mediante un transformador de corriente 220/5 V y 2.5 A de intensidad.

Sistema operativo

Como sistema operativo se utilizará **Raspbian**. Es una distribución del sistema operativo GNU/Linux basado en Debian y optimizado para la CPU de RPi. Lo más característico del SO son los cálculos optimizados en coma flotante por hardware y el menú personalizado de configuración *"raspi-config"*. Dadas las especificaciones técnicas de cada modelo de RPi utilizado, se escogerán diferentes versiones para conseguir un procesado más eficiente.

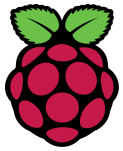


Figura 33: Raspbian

- **Raspbian desktop**. Para las Raspberrt Pi 3. Incluye una capa gráfica que hace más fluido el trabajo de gestión de datos y programación
- **Raspbian lite**. Para las Raspberrt Pi Zero. Carece de capa gráfica por lo que requiere de menos recursos de CPU

Configuración OTG

OTG significa *"On The Go"* debido a que permite una conexión rápida y sencilla entre dispositivos. Es necesario que los dispositivos conectados tengan los drivers pertinentes para poder entender la información transmitida. En este caso se conectarán las dos RPi Zeros a la principal y esta la reconocerá y registrará como un dispositivo externo.

En primer lugar se deberá configurar en todos los dispositivos la conexión OTG. Para ello se editará dos archivos,

- ***config.txt***. Con ello se carga el driver *dwc2*, recomendado para transferencia de archivos. En este caso hay que añadir una nueva línea al final: `dtoverlay=dwc2`.
- ***cmdline.txt***. Habrá que configurar la dirección local y remota para la comunicación OTG: `modules-load=dwc2,gether, g_ether.host_addr=00:22:82:ff:ff:01, g_ether.dev_addr=00:22:82:ff:ff:11`.

Cada vez que se inicia una RPI zero conectada a OTG, se crearán dos interfaces de red nuevas. Una en el host que se conectará en este caso a la Pi 3 (Recordar que se está hablando desde una Pi Zero) y otra en lA Pi Zero misma. Todas los interfaces de red necesitan una MAC única para que el dispositivo sea identificable. Por ejemplo, cuando se realiza una conexión a un router, este comprueba la MAC del dispositivo y después asigna una IP. El problema es que cada vez que una RPi se conecta, genera automáticamente una MAC aleatoria. Añadiendo el parámetro `gether.devaddr` se asigna una MAC fija al interfaz de red de la RPI zero. Por otro lado, `g_ether.hostaddr` permite asignar una MAC al interfaz de red de la RPi 3 que será necesaria para más adelante.

Pi Zero: Cambiar el nombre del Host

La conexión *ssh* se puede realizar a través del nombre del host, en este acso la RPi Zero. Por defecto el nombre de todas las raspberries es *raspberry* por lo que si no se cambia, se estará conectando dos dispositivos con el mismo nombre a la vez, causando un conflicto.

Para cambiar el nombre de los host en la lista se deberán editar dos archivos,*"/etc/hosts"* y *"/etc/hostnames"*. En este proyecto se llamarán a las dos Pi

zeros, "Usb1" y "Usb2". Posteriormente habrá que reiniciarlas para que los cambios se hagan efectivos.

Pi Zero: Definir una IP estática

Por defecto ciene configurada una IP dinámica que se asigna aleatoriamente cada vez que se inicia la RPi. Para facilitar las labores de conexión es conveniente asignar una IP estática en cada una de las Pi Zeros.
Se editará el archivo *"/etc/dhcpcd.conf"*. En él habrá que añadir la interfaz de red deseada con su correspondiente IP:

```
interface usb0
static ip_address=10.0.11.2
```

Cada Pi Zero deberá tener una IP única, en este caso las IPs asignadas son *10.0.11.2* y *10.0.12.2* Una cosa a tener en cuenta es que sólo está modificando el tercer número de la IP. Esto es debido a que las direcciones IP se desglosan en subredes.

Por ejemplo, 10.0.11.0 en este caso subred de USB 1 es añadida a una tabla de IPs que será consultada cuando el propio dispositivo con IP 10.0.11.1, o .2 se conecte para saber a que subred pertenece.
Esto es un problema cuando se conectan dos RPis en la misma subred dado que cuando se hace *ping* para comprobar la conexión, esta sólo detectará el primer dispositivo que aparezca en la lista, haciendo referencia a el ejemplo, detectará únicamente el dispositivo on IP 10.0.11.1. Por lo tanto, la manera más sencilla de solucionar el problema es crear las dos dubredes tál y como se ha indicado.

Pi 3: Reglas UDEV

Cada vez que una PI Zero se conecta al host, el host crea una interfaz de red que se llamará USB0 ,USB1, USB2... a medida que vaya aumentando el número de

conexiones.En estos interfaces queda registrada la IP del dispositivo. El problema es que estos interfaces son reutilizados cuando un dispositivoo se desconectan y posteriormente otro similar con otra IP se conecta. Esto resulta en que el segundo dispositivo conectado no será reconocido correctamente puesto que la IP del mismo y la del interfaz de red no coinciden.

La mejor forma de arreglarlo es utilizar las **reglas de UDEV** para cambiar el nombre de los interfaces de red basándose en la dirección MAC del cliente.

Para cumpliralas se deberá crear el archivo
"/etc/udev/rules.d/90-panocluster.rules"
y añadir las siguiente líneas de código.

```
SUBSYSTEM=="net", ATTRaddress=="00:22:82:ff:ff:01", NAME="ethpi1"
SUBSYSTEM=="net", ATTRaddress=="00:22:82:ff:ff:02", NAME="ethpi2"
```

De esta manera, cada vez que se conecte una PI Zero, antes de activar el interfaz de red, comprobará su dirección MAC e indicará que interfaz asignado

Pi 3: Configurar las interfaces de red con IP estática

Una vez que la reglas de UDEV asignan el interfaz se pueden asignar las IPs a reconocer. Editando el archivo *"/etc/dhcpcd.conf"* podremos crear los interfaces de red necesarios con las IPs estáticas pertinentes. Se añadirán las siguiente líneas.

```
interface ethpi1
static ip_address=10.0.11.1/24

interface ethpi1
static ip_address=10.0.12.1/24
```

Ahora todas las RPi Zero serán reconocidas por el Host.
Un esquemático sencillo de como quedaría la conexión:

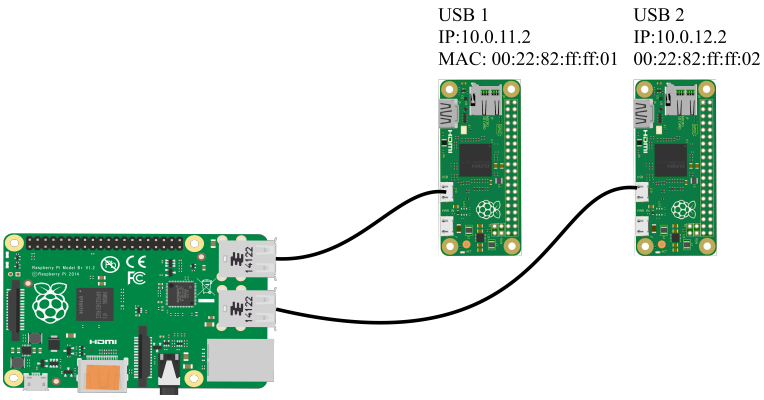


Figura 34: Esquema de montaje

4.12.3. Transferencia de información entre Raspberries

Una vez implementado todo lo anterior, la RPi 3 reconoce sin ningún tipo de problema ambas Pi Zeros. Aún así es necesario implementar en software el algoritmo que permita la transferencia automática de datos.

Existirá una doble transferencia de datos. Por un lado desde las Pi Zeros a la Pi 3, y por otro, cubierto por ROS, desde la RPi 3 a el ordenador.
Ambas se llevarán a cabo a través de medios diferentes. En el primer caso se realiará mediante cable USB puesto que las Pi Zero no tienen sensor Wifi. En el segundo se realizará a través de Wifi. En cualquier caso, se utilizará el protocolo **SSH** para la transmisión.

Protocolo SSH (*Secure SHell*) Es un protocolo que permite, mediante una estructura cleinte/servidor, las comunicaciones seguras entre dos sistemas y el acceso remoto a diferentes hosts. Surgío para sustituir conexiones menos seguras como *telnet* o *rsh*. SSH encripta la sesión de conexión de manera que es imposible que alguien ajeno pueda obtener acceso a la comunicación. Además permite copiar datos de un equipo a otro a través de sesiones FTP cifradas.

En este proyecto se necesita que todas estas comunicaciones se realicen de manera automática. Esto se hará mediante código en Python en el que se utilizará la

librería *Paramiko*. No es la librería más eficiente, pero sí la que mejor compatibilidad ofrece con todo tipo de SO.

Comunicación RPi 3 - RPi Zeros

En este caso hay que establecer una doble comunicación para cada Pi Zero, a demás esta debe ser en paralelo y en un lapso de tiempo lo menor posible para permitir que el ratio de fps sea lo mayor posible. Para ello se han implementado tres funciones:

- Conexión. Esta función genera los objetos que posteriormente se utilizarán para la transferencia de comandos o archivos.

```
def ssh_client():
   ip='rpimain'
   print ("Connecting to: RPimain")
   ssh_client=paramiko.SSHClient()
   ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
   ssh_client.connect(hostname=ip,username='pi',password='raspberry')
   ftp_client=ssh_client.open_sftp()
   stdin,stdout,stderr=ssh_client.exec_command('echo Connection_OK')
   print (stdout.read())
   return (ssh_client,ftp_client)
```

- Photo. Esta función ordenará a la RPi Zero tomar un foto y transerirla a la RPi 3.

```
def photo(ssh_client,ftp_client):
   ftp_client.get('Desktop/foo21.jpg','foo21.jpg')
```

- Close. Cierra la conexión ftp a partir del objeto de conexión ftp.

```
def Close(ftp_client):
    ftp_client.close()
    print('Connection finished')
    return(0)
```

Comunicación PC - RPi 3

Las funciones son prácticamente iguales, la única diferencia es que en este caso no sera necesario enviar el comando de adquisición de foto y simplemente habrá que copiarla.

4.12.4. Cámaras

La cámara es un elemento esencial en el desarrollo de este proyecto. Su correcta implantación y calibración serán determinantes en el éxito del mismo. Por coste y prestaciones se han escogido unas cámaras con las siguientes especificaciones:

- **Sensor:** OV5647 5 Megapíxeles
- **Apertura (F):** 1,8
- **Ángulo Diagonal:** 60 grados

Instalación y calibración de las cámaras

Para la adquisición de las imágenes es imprescindible la correcta instalación de las cámaras. En las imágenes stereo hay muchos parámetros que influyen y pueden causar que el post-procesamiento sea incorrecto.
Entre estos parámetros están aquellos que dependen de cómo se haya tomado la foto, y los que dependen únicamente de la cámara y sus características.

En el proceso de calibración se trata de eliminar todas aquellas interferencias causadas por las diferentes variables. Para saber contrarrestarlas es necesario conocer qué son.

Existen dos tipos de distorsión que dependen de cómo y en que posición se hayan tomado las fotos. Estas son la radial y la tangencial.

Radial. La distorsión radial causa que las líneas rectas aparezcan curvas. Este efecto se exagera a medida que los objetos no aparecen más lejos del centro de la imagen. Se puede apreciar este efecto en una de las imágenes tomadas para la calibración de la camara.

Figura 35: Efecto de la distorsión radial

La distorsión radial se soluciona mediante la siguiente transformación.

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{13}$$

$$y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{14}$$

Tangencial. Esta distorsión se debe a que la lente que toma la imagen no está alineada perfectamente paralela a el plano de la imagen. Esto puede causar un falso efecto de cercanía del contenido de la imagen.
Se puede corregir mediante la siguiente transformación.

$$x_{corregida} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{15}$$

$$y_{corregida} = y + [2p_2 xy + p_1(r^2 + 2y^2)] \tag{16}$$

En resumen, para poder aplicar las ecuaciones de transformación se necesitan cinco parámetros conocidos como los coeficientes de distorsión que son:

$$Coeficientesdedistorsion = (k_1 k_2 p_1 p_2 k_3) \tag{17}$$

A demás, se necesita mas información específica de cada cámara. Como la longitud focal (f_x, f_y) y centros ópticos (c_x, c_y) . Esta información es conocida también como la matriz de cámara. Depende únicamente de la cámara y no de cómo o en que posiciónse halla tomado las imágenes. Una vez calculados no varían y son siempre los mismos.

$$Matriz = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{18}$$

Tál y como se ha indicado antes, para aplicaciones de visión stereo es necesario corregir estas imperfecciones antes. Para ello habra que tomar una serie de imágenes que permitan obtener los coeficientes indicados.

En estas imágenes se tratará de identificar puntos que faciliten el cálculo y para ello se utiliza la plantilla del tablero de ajedrez. Esta plantilla de cuadrados blancos y negros facilita la identificación de los mismos puntos y rectas en diferentes imágenes. Para este proyecto se ha utilizado una plantilla de 9x6 y con unas dimensiones de cuadrados de 25x25 mm que es la más común. Estos parámetross son importantes puesto de cara a la programación simplifica mucho el proceso conocer la cantidad de puntos a buscar de antemano.

(9x6 se corresponde con los puntos que se buscan y no con la cantidad de cuadrados. Los puntos objetivo son aquellos donde concurren cuatro aristas de cuadrados)

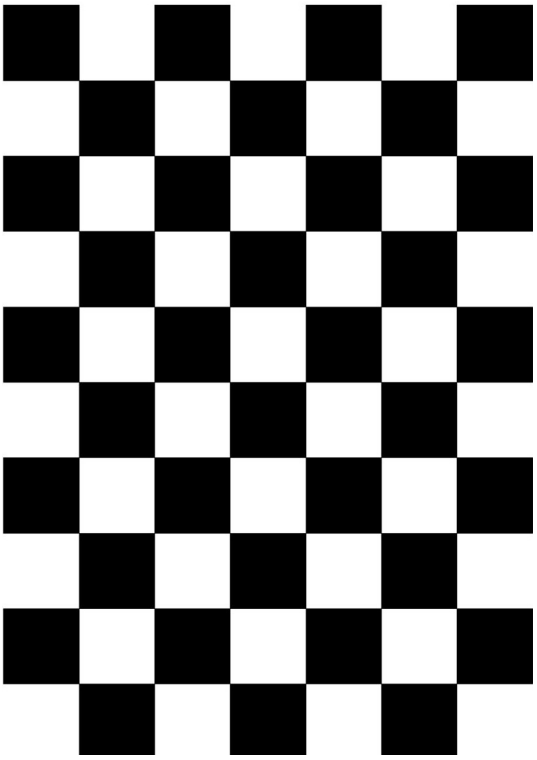


Figura 36: Plantilla utilizada para la calibración 9x6

Calibración

OpenCV tiene librerías especificas de calibración que facilitan enormemente el proceso. De hecho es casi un proceso automático en el que tan solo hay que ordenar los pasos a realizar. En primer lugar se ha de detectar el patrón de puntos de la plantilla de varias imágenes para poder contrastar.

```
ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
if ret == True:
   corners2 = cv2.cornerSubPix(img,corners,(11,11),(-1,-1),crit)
   imgpoints.append(corners2)
   img = cv2.drawChessboardCorners(img, (7,6), corners2,ret)
   cv2.imshow('img',img)
   cv2.waitKey(500)
```

A esta función se obtiene la siguiente imagen como salida.

Figura 37: Salida a la funcion de detección de puntos

Una vez se tienen los puntos objetivo se puede proceder a la función de calibración, que obtiene todos los coeficientes deseados. Estos coeficientes serán guardados en un archivo .txt para posteriormente poder realizar el procesamiento de la calibración mediante hardware.

```
datos = cv2.calibrateCamera(objpoints, imgpoints,img.shape[::-1],None,None)
```

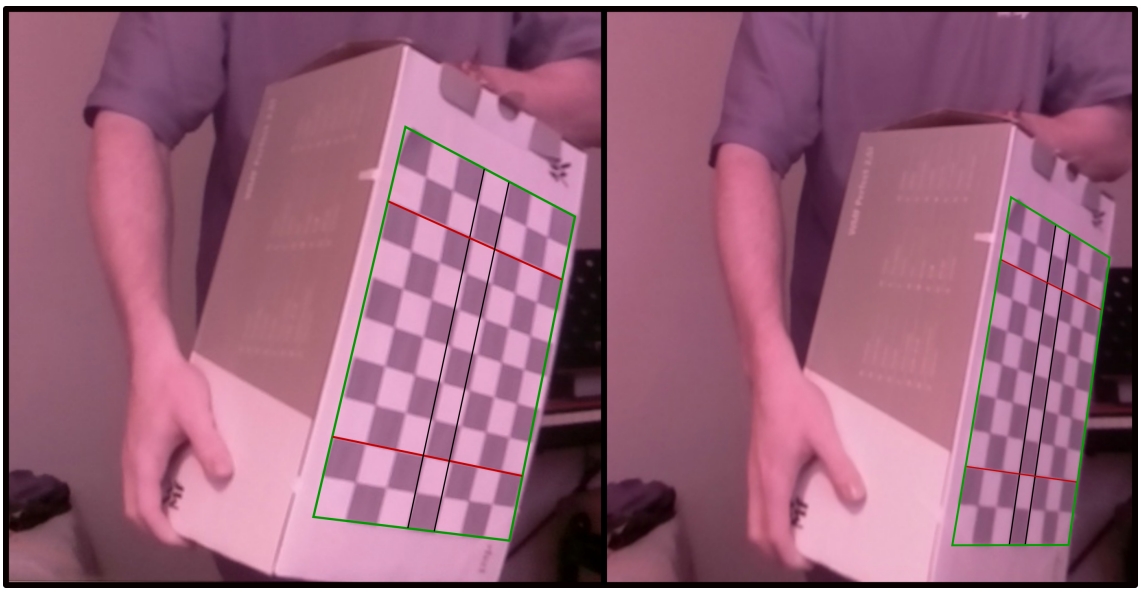


Figura 38: Imagen sin calibrar vs Imagen calibrada

Los resultados de los coeficientes de calibración para la cámara derecha se muestran a continuación.

$$Distorsion = (-4.27e-01, 2.82e-01, -7.10e-03, -3.95e-03, -1.56e-01) \quad (19)$$

$$Matriz = \begin{bmatrix} 6.20e+02 & 0 & 3.25e+02 \\ 0 & 6.22e+02 & 2.49e+02 \\ 0 & 0 & 1 \end{bmatrix} \quad (20)$$

4.13. Abstracción artística

Desde nuestros primeros pasos como especie hemos luchado por comprender y dominar la realidad que nos rodea. Fruto de ese esfuerzo hemos generado unas interpretaciones de la misma con sus consecuentes soluciones. Durante la mayor parte nuestra existencia la compresión, solución e interpretación del universo que nos rodea se han considerado como una misma actividad. Sin embargo, a partir de los movimientos artísticos del siglo XVIII, el racionalismo o el surgimiento de las academias, con el objetivo de ampliar nuestra comprensión, dominio y facetas trabajadas de la realidad, dividimos esta actividad en diferentes especialidades: comprensión, ciencia; solución, tecnología; interpretación, arte.

En la actualidad corremos el riesgo de desgajar demasiado estas especialidades. Nos olvidamos de que los grandes artistas, filósofos, científicos e ingenieros de la historia compartían estas facetas: Pitágoras, Aristóteles, Averroes, Avicena, Descartes, ... Nos olvidamos de que es necesario dominar la simetría, la perspectiva, la luz y la composición de los colores para hablar de arte pictórico; o de que para la composición musical artística es vital el conocimiento de la formación, propagación y recepción del sonido.

Con el fin de reivindicar como irrenunciable esta unión, no presento el proyecto exclusivamente como un sistema diseñado para ser montado y utilizado en un dron sino como la expresión de la inteligencia humana que afronta los problemas que el ser humano se encuentra al dominar la realidad, es decir, como ingeniería y que la interpreta estéticamente, es decir, como arte.

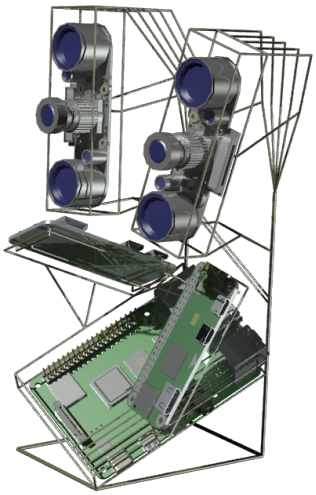


Figura 39: Montaje 3D

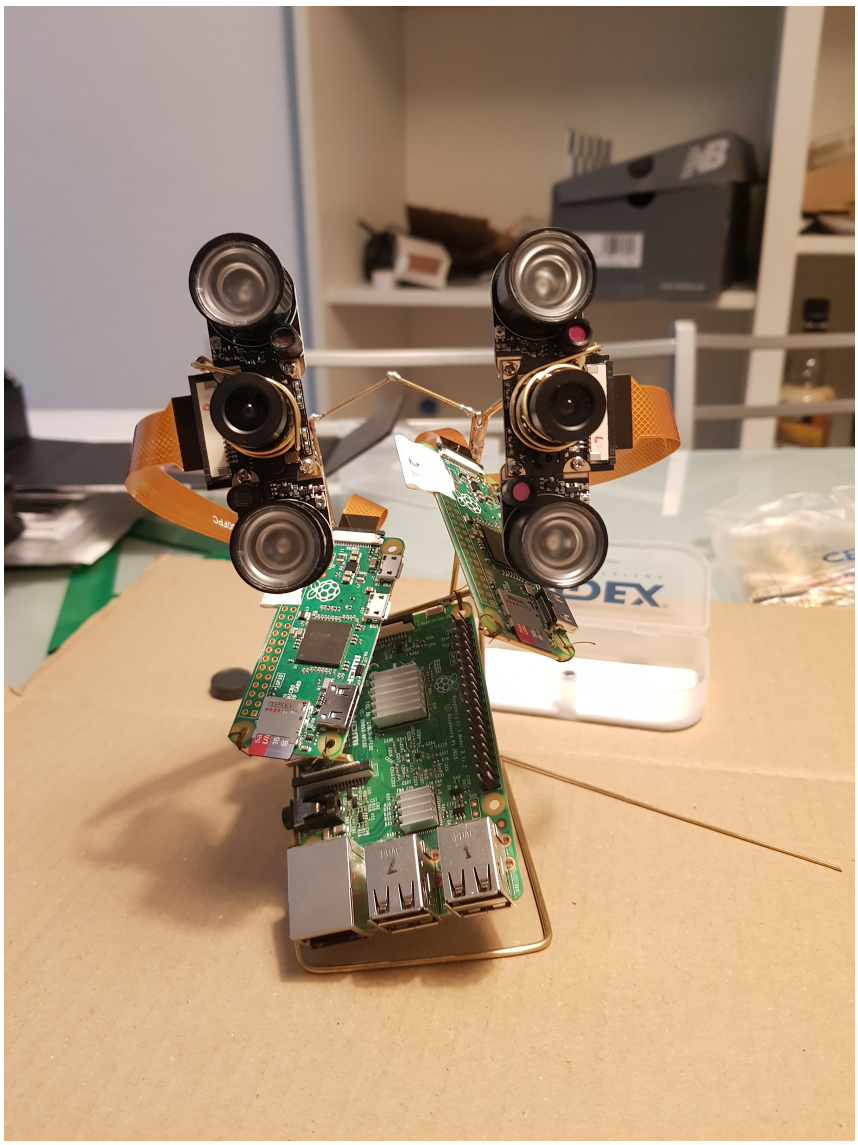


Figura 40: Montaje final

4.14. Conclusiones

De acuerdo con los objetivos específicos, se considera que las imágenes sintéticas obtenidas de la combinación de *Unreal Engine* y *Airsim*, junto a los diferentes algoritmos para la extracción del emphground truth de manera automática (Centro de masas, Regresión y Sectorización) permiten el entrenamiento de redes neuronales para el guiado de vehículos autónomos.

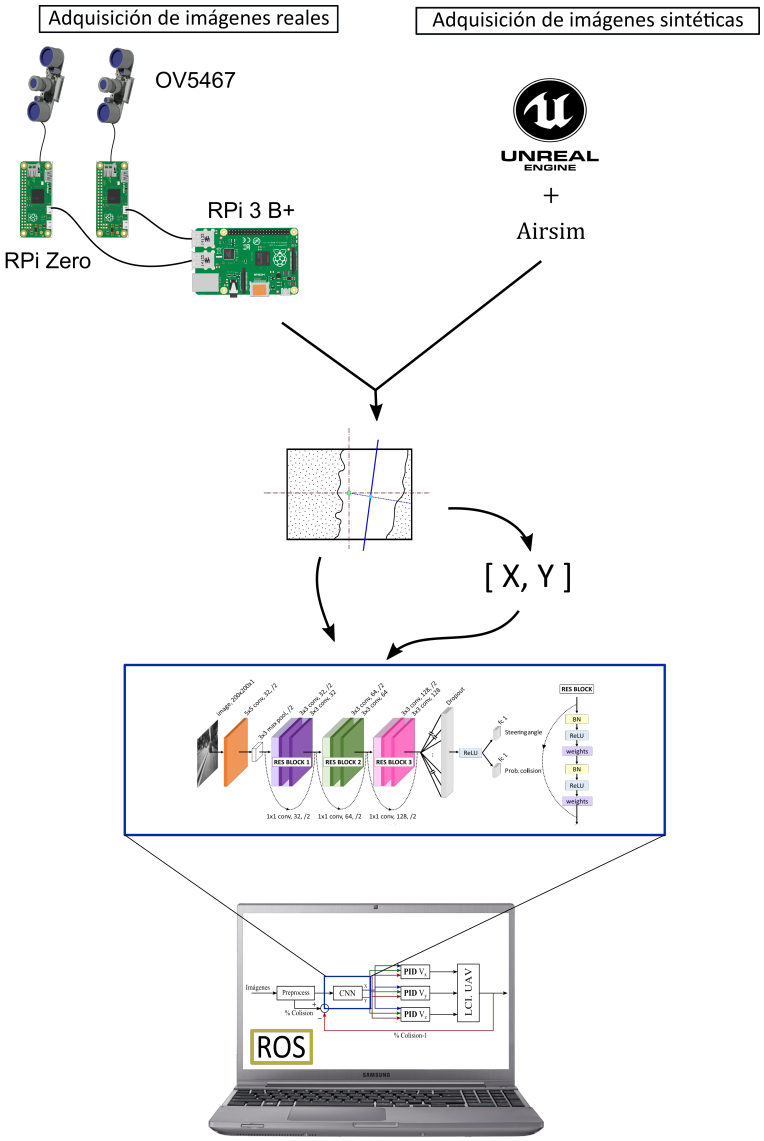


Figura 41: Utilidad del proyecto

Concretamente, el algoritmo óptimo para la generación de *ground truth* es el de regresión dado que presenta un RMSE inferior a todos los demás. Tras el entrenamiento, de acuerdo a la gráfica de error, se escoge la etapa 49 como la óptima antes de que el entrenamiento empiece a mostrar comportamientos erróneos.

Posteriormente se ha implementado la arquitectura de control que es capaz de transformar la salida de la red neuronal en referencias de control para el guiado del UAV. Con ella se ha comprobado la versatilidad de ROS a la hora de tratar información en aplicaciones de tiempo real.

Por último se ha conseguido implementar una solución hardware capaz de adquirir imágenes stereo tras un proceso de calibración previo. No obstante se considera que la robustez de esta solución debe ser mejorada, ya que el entorno de los UAV es bastante agresivo tanto por condiciones climatológicas como por posibles impactos.

4.14.1. Trabajos futuros

Para conseguir operar un drone con la información generada en este trabajo, se deberá, a modo de paso intermedio, implementar otro nodo capaz de transformar las referencias de control generadas en la arquitectura, en señales de radio que serán enviadas desde el PC al UAV.

Este nodo existe actualmente: utiliza tecnología Arduino junto a un transmisor de radio *Turnigy 9X* .De este modo se podrá depurar todo el sistema de manera mucho más ágil que si se implementara directamente en el prototipo hardware. Si se obviara este paso, sería necesario llevar a cabo vuelos completamente autónomos para observar el funcionamiento. Este paso intermedio permitirá realizar más vuelos experimentales que aporten información de más calidad y más útil.

Finalmente con el fin de implementar todo el sistema final, se deberá rediseñar la disposición de la solución hardware para mejorar sus propiedades aerodinámicas y trabajar en la comunicación entre Raspberry y la controladora del UAV.

ESQUIVACIÓN DE OBJETOS EN VEHÍCULOS AUTÓNOMOS MEDIANTE TÉCNICAS DE *MACHINE LEARNING*

PRESUPUESTO

5. Presupuesto

5.1. Cuadro de medidas

C01.: Materiales electrónicos del hardware

Código	Descripción	Cantidad
C01.1	Rapberry Pi 3 B+	1
C01.2	Rapberry Pi Zero	2
C01.3	Led smd 0603	10
C01.4	Camera OV5647 5Mpx	2
C01.5	Tarjeta de memoria microSD	2
C01.4	Bus CSI	2
C01.4	Cable USB-Micro USB	2

C02.: Materiales necesarios para la configuración del Hardware

Código	Descripción	Cantidad
C02.1	Teclado y ratón inalámbrico USB	1
C02.2	Extensor de puertos USB 1-4	1
C02.3	Adaptador micro-HDMI/HDMI	1
C02.4	Conversor USB/micro-SD	1
C02.5	Adaptador micro-USB/USB-C	1
C02.6	Cable Micro-USB/RJ-45	1

C03.: Materiales para la fabricación

Código	Descripción	Cantidad
C03.1	Soldador	1
C03.2	Estaño 100 g	2
C03.3	Varilla de latón 0.8 mm	2
C03.4	Varilla de latón 1.0 mm	2
C03.5	Varilla de latón 1.2 mm	2
C03.6	Papel de lija grano medio 280	1
C03.7	Flux 100 ml	1

C04.: Mano de obra

Código	Descripción	Cantidad
C04.1	Horas de investigación	502
C04.2	Horas de diseño	53
C04.3	Horas de montaje	156
C04.4	Horas de programación	249

5.2. Cuadro de precios

C01.: Materiales electrónicos del hardware

Código	Descripción	Precio(€)
C01.1	Rapberry Pi 3 B+	34,99
C01.2	Rapberry Pi Zero	13.67
C01.3	Led smd 0603	0,14
C01.4	Camera OV5647 5Mpx	15,43
C01.5	Tarjeta de memoria microSD	8,50
C01.4	Bus CSI	1,61
C01.4	Cable USB-Micro USB	3,50

C02.: Materiales necesarios para la configuración del Hardware

Código	Descripción	Precio (€)
C02.1	Teclado y ratón inalámbrico USB	19,99
C02.2	Extensor de puertos USB 1-4	4,50
C02.3	Adaptador micro-HDMI/HDMI	4,99
C02.4	Conversor USB/micro-SD	6,99
C02.5	Transformador 220AC/5DC	12,50
C02.6	Cable Micro-USB/RJ-45	4,50

C03.: Materiales para la fabricación

Código	Descripción	Precio (€)
C03.1	Soldador	13,45
C03.2	Estaño 100 g	3,50
C03.3	Varilla de latón 0.8 mm	1,00
C03.4	Varilla de latón 1.0 mm	1,40
C03.5	Varilla de latón 1.2 mm	1,80
C03.6	Papel de lija grano medio 280	1,50
C03.7	Flux 100 ml	5,59

C04.: Mano de obra

Código	Descripción	Precio (€)
C04.1	Horas de investigación	47
C04.2	Horas de diseño	42
C04.3	Horas de montaje	20
C04.4	Horas de programación	47

5.3. Presupuesto

C01.: Materiales electrónicos del hardware

Código	Descripción	Cantidad	Precio (€)	Importe
C01.1	Rapberry Pi 3 B+	1	34,99	34,99
C01.2	Rapberry Pi Zero	2	13,67	27,34
C01.3	Led smd 0603	10	0,14	1,4
C01.4	Camera OV5647 5Mpx	2	15,43	30,86
C01.5	Tarjeta de memoria microSD	2	8,50	17
C01.4	Bus CSI	2	1,61	3,22
C01.4	Cable USB-Micro USB	2	3,50	7
			Total capítulo	121,81

C02.: Materiales necesarios para la configuración del Hardware

Código	Descripción	Cantidad	Precio	Importe
C02.1	Teclado y ratón inalámbrico USB	1	19,99	19,99
C02.2	Extensor de puertos USB 1-4	1	4,50	4,50
C02.3	Adaptador micro-HDMI/HDMI	1	4,99	4,99
C02.4	Conversor USB/micro-SD	1	6,99	6,99
C02.5	Transformador 220AC/5DC	1	12,50	12,50
C02.6	Cable Micro-USB/RJ-45	1	4,50	4,50
			Total capítulo	53,47

C03.: Materiales para la fabricación

Código	Descripción	Cantidad	Precio	Importe
C03.1	Soldador	1	13,45	13,45
C03.2	Estaño 100 g	2	3,50	7,00
C03.3	Varilla de latón 0.8 mm	2	1,00	2,00
C03.4	Varilla de latón 1.0 mm	2	1,40	2,80
C03.5	Varilla de latón 1.2 mm	2	1,80	3,60
C03.6	Papel de lija grano medio 280	1	1,50	1,50
C03.7	Flux 100 ml	1	5,59	5,59
			Total capítulo	35,94

C04.: Mano de obra

Código	Descripción	Cantidad	Precio	Importe
C04.1	Horas de investigación	502	47	23.594
C04.2	Horas de diseño	53	42	2.226
C04.3	Horas de montaje	156	20	3.120
C04.4	Horas de programación	249	47	11.703
			Total capítulo	40.643

5.4. Resumen del preupuesto

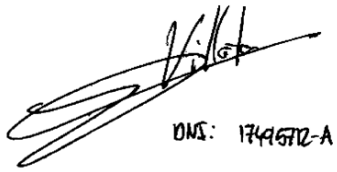
Capítulo Resumen		Euros	%
C01	Materiales electrónicos del hardware	121,81	23,594
C02	Materiales necesarios para la configuración del Hardware	53,47	2,226
C03	Materiales para la fabricación	35,94	3,120
C04	Mano de obra	40.643	11,703
	Total	40.854	100 %

Total ejecución Material	40.854
13 % Gastos generales	5.311,02
6 % Beneficio industrial	2.451,24
SUMA DE GG y BI	7.762,26
21 % de IVA	8.579,34
TOTAL PRESUPUESTO GENERAL	51.192,60 €

Asciende el presupuesto general a la expresada cantidad de CINCUENTA Y UN MIL CIENTO NOVENTA Y DOS EUROS con SESENTA CÉNTIMOS

FIRMADO:

D. David Villota Miranda



Logroño a 18 de Febrero de 2019

ESQUIVACIÓN DE OBJETOS EN VEHÍCULOS AUTÓNOMOS MEDIANTE TÉCNICAS DE *MACHINE LEARNING*

PLIEGO DE CONDICIONES

.

6. Pliego de condiciones

6.1. Introducción al pliego de condiciones

El autor de este proyecto ha cursado los estudios de Máster en Ingeniería Industrial de La Rioja, cumpliendo con la normativa establecida por la Escuela Superior de Ingeniería Industrial en la normativa de trabajo fin de máster.

El objeto de este Pliego de Condiciones es recoger y establecer todas las disposiciones técnicas, administrativas y económicas, y las normativas que ha de regir este proyecto.

El diseño de este proyecto y sus características han sido descritos en detalle en la memoria del proyecto y sus anexos.

Las condiciones que se especifican en este documento tratan de cumplir con la calidad esperada para este proyecto. En caso de no realizarse según estas condiciones, el proyectista no se responsabilizará de los fallos o averías que puedan ocasionarse en su funcionamiento, y los problemas derivados repercutirían sobre terceras personas.

Todas las modificaciones que puedan sufrir el proyecto y sus documentos deberán ser aprobadas por el ingeniero o proyectista.

6.2. Condiciones Generales

Este proyecto se ajusta a los reglamentos y normativas electrónicas vigentes. Una vez terminado el proyecto se podrán llevar a cabo modificaciones pero siempre bajo la supervisión del proyectista.

La propiedad intelectual del autor y director del Trabajo Fin de Grado se regirá por el Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual.

6.3. Condiciones Administrativas

El proyecto constará de los siguientes documentos:

- Un Índice General que indicará la página de comienzo de cada uno de los documentos que forman el proyecto.
- Una Memoria donde se considerarán las necesidades a satisfacer y los factores técnicos a tener en cuenta entrando en profundidad en las posibles soluciones técnicas y en la justificación de la solución elegida.
- Anexos donde se recogerá la documentación considerada de interés para ampliar la descripción detallada de los componentes del sistema.
- Pliego de Condiciones, este documento en el que se establecen las diferentes condiciones técnicas, económicas y administrativas para que proyecto pueda materializarse, evitando posibles malinterpretaciones.
- Presupuesto donde se recogerá el coste de todos los componentes utilizados y la suma total que, junto a la mano de obra, dará el coste final del proyecto. Dicho presupuesto contiene la valoración económica global, desglosada y ordenada por partidas.

6.4. Normativa y Reglamentación

El proyecto estará regido tanto por la normativa española como por la internacional:

Reglamento relacionado a productos electrónicos

Respecto al desarrollo de productos electrónicos, se pueden encontrar en AENOR (Asociación Española de Normalización y Certificación) las siguientes normativas:

- Norma UNE1302—2:1973 Vocabulario electrotécnico. Electrónica.
- Norma UNE-EN611000-4-3-1998. Compatibilidad electromagnética.

Este proyecto debido a sus características se rige bajo el reglamento de Baja Tensión: *"Se calificará como instalación eléctrica de baja tensión todo conjunto los aparatos y de circuitos asociados en previsión de un fin particular: producción, conversión, transformación, transmisión, distribución o utilización de la energía eléctrica, cuyas tensiones nominales sean iguales o inferiores a 1000 V para corriente alterna y 1500 V para corriente continua."*

Normativa relacionada con materiales y equipos

Los materiales y equipos de este proyecto deben cumplir los estándares nacionales e internacionales en vigor y obligado cumplimiento. Entre otros:

- UNE 20-324 Grados de protección de los envolventes del material eléctrico de baja tensión.
- UNE 20-334 Conductos para instalaciones eléctricas.
- UNE 21-401 Conductores eléctricos aislados.
- UNE 21-402 Conductores eléctricos aislados y desnudos.
- REAL DECRETO 552/2014 Regulación de la utilización civil de aeronaves pilotadas por control remoto

6.5. Condiciones facultativas

Dirección

La dirección del montaje del será llevada a cabo, en su totalidad por el ingeniero proyectista, o por cualquier otra persona en la que éste delegue, atendiendo a la capacidad de dicha persona para responsabilizarse de dicha dirección.

Una vez realizada la instalación, ésta podrá ser utilizada por cualquier persona con conocimientos demostrables suficientes sobre el sistema y sus componentes, las tecnologías implicadas y el funcionamiento global y, por partes, del sistema.

En caso de avería o pérdida de información por una utilización incorrecta, el ingeniero proyectista o la persona en la que haya delegado la dirección del proyecto, quedan exentas de responsabilidad.

Libro de órdenes

El montaje e instalación de todos los elementos que componen el proyecto se realizará atendiendo al siguiente orden de prioridad en caso de que haya alguna contradicción:

- Presupuesto.
- Pliego de condiciones .
- Memoria

Este libro de órdenes debe estar conforme al Decreto 462/1971 de 11 de marzo, y la Orden de 9 de junio de 1971.

Modificaciones

Si fuera necesario realizar alguna modificación en el presente proyecto, deberá comunicarse con anterioridad a su realización al Ingeniero Director, quién deberá dar la correspondiente autorización.

En caso de realizarse modificaciones en la instalación que no hayan sido previamente comunicadas y autorizadas por el ingeniero Director, las consecuencias que dichos cambios puedan ocasionar serán de total responsabilidad del instalador que las realice.

Respecto a los cambios en la instalación realizados por el propietario de la misma, no serán tratados de forma especial y, en ningún caso, quedan eximidos de la autorización del ingeniero Director.

6.6. Condiciones de materiales y equipos

A continuación se detallan las condiciones tanto de hardware como de software con las que hay que contar para hacer uso de la aplicación del proyecto.

Condiciones técnicas de los materiales

Todos los materiales y componentes, utilizados en el proyecto, deben cumplir todas las especificaciones técnicas que aparecen descritas en la Memoria y deben cumplir asimismo todas las normas descritas en este pliego de condiciones.

Si se considera necesario reemplazar algún componente o material por otro, los nuevos deberán tener las mismas características que los reemplazados, inhibiéndose el ingeniero proyectista de cualquier responsabilidad por fallo, si no se cumplen estos requisitos.

Para el desarrollo e implementación del proyecto se deberá disponer de un PC que será el encargado de recoger los datos, analizarlos y procesarlos. Para ello deberá disponer al menos del siguiente software:

- Python IDLE
- ROS
- Tensorflow
- Keras
- Unreal Engine
- Inkscape

6.7. Condiciones económicas

Errores en el proyecto

En el caso de existir algún tipo de error en el proyecto se avisará inmediatamente al proyectista y se le informará con detalle de los errores encontrados.

Además se dejará de usar la aplicación hasta que los errores queden solucionados para prevenir cualquier tipo de daño.

Liquidación

Terminado la fase de pruebas del UAV, se procederá a la liquidación final, en la que se incluye el importe de las unidades de realización, así como las posibles modificaciones del proyecto que hayan sido aprobadas por la dirección del proyecto.

Al suscribir el contrato, el cliente habrá de abonar el 80 % del presupuesto. El 20 % quedará como garantía durante los seis primeros meses, a partir de la fecha de puesta en marcha del sistema.

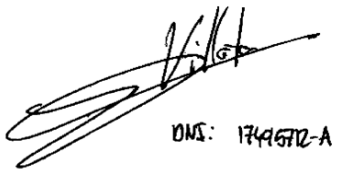
Si transcurridos seis meses desde la puesta en marcha no se ha manifestado ningún defecto o error de funcionamiento, el cliente abonará el 20 % que estaba pendiente. A partir de ese momento, se considerarán concluidos los compromisos entre ambas partes, a excepción del periodo de garantía.

6.8. Disposición final

Las partes contratantes, tanto la dirección del proyecto como la empresa cliente, se ratifican en el contenido del presente pliego de condiciones, que tiene igual validez, a todos los efectos, que una escritura pública, prometiendo su fiel cumplimiento.

FIRMADO:

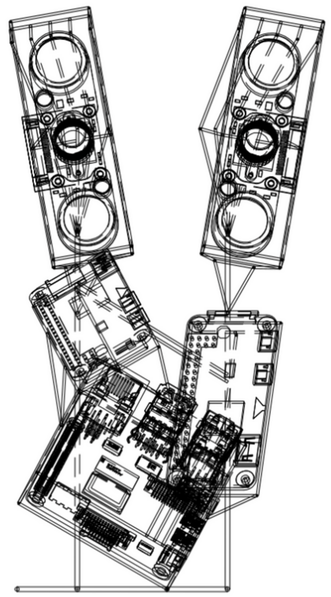
D. David Villota Miranda

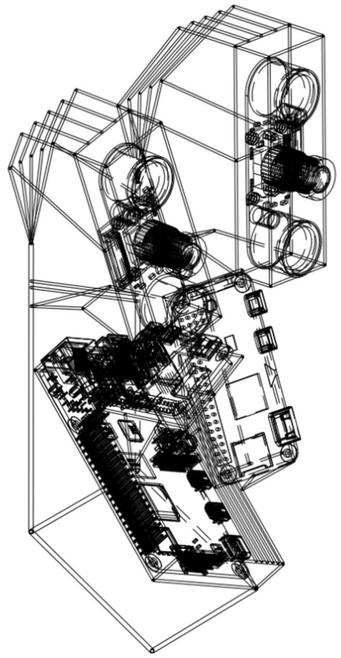


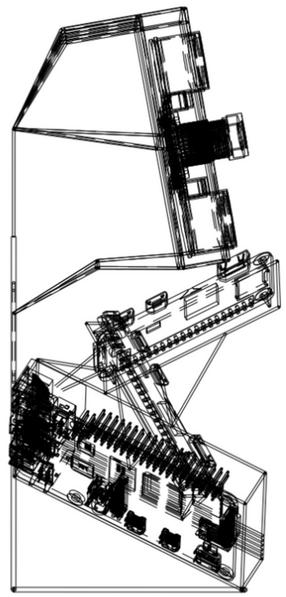
.

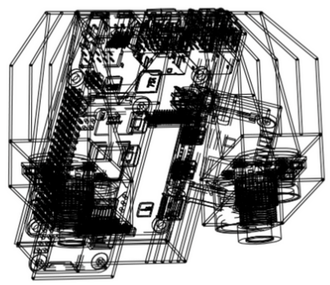
ESQUIVACIÓN DE OBJETOS EN VEHÍCULOS AUTÓNOMOS MEDIANTE TÉCNICAS DE *MACHINE LEARNING*

7. Planos









	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL
Dibujado	27/08/2019	D. Villota Miranda		*Universidad de La Rioja*
Comprob.	31/01/2020	D.Villota Miranda		MÁSTER EN INGENIERÍA INDUSTRIAL
D.S.Normas	U.N.E	Tolerancia general		

PROYECCIÓN	TRABAJO FIN DE MÁSTER	Número: 1
	PROTOTIPO DE MONTAJE HARDWARE	REFERENCIA: P.1
		Sustituye a: --
		Sustituido por: --